(continued from part 19)

How does MOS ROM work?

Let's take a detailed look at the type of read only memory that is used in calculators. This is made with p-channel MOSFETs, like a calculator's main chip.

Figure 5 shows four 1-bit memory cells in the lower left-hand corner of a ROM array that could be any size. Each cell is a simple MOS transistor if a 1 is stored. If a 0 is to be stored, the cell is made from an incomplete transistor – without a gate – so it can never be turned on.

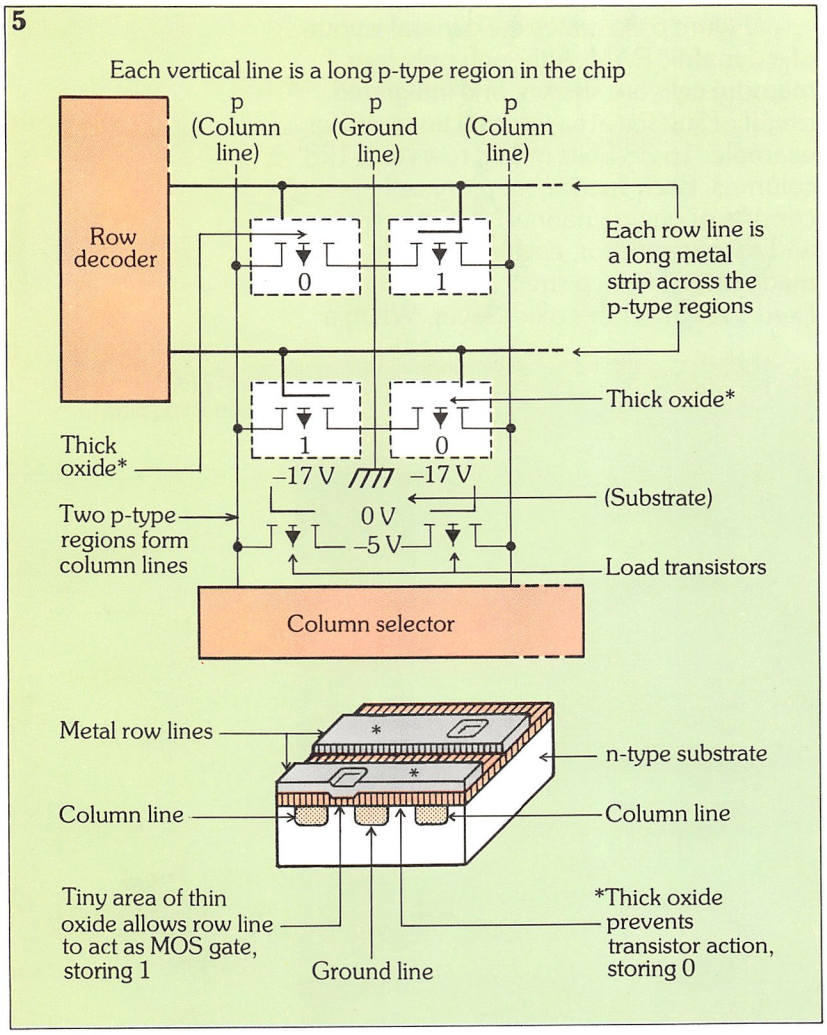
The row decoder transmits a 'low' voltage (−5 V) in the selected row line. This switches on all the transistors with a gate, i.e. all those that represent a stored 1 in that row. Each 1 is then transmitted in the column lines to the column selector as 0 V. The incomplete transistor (representing 0s) will, of course, not turn on, and the relevant column lines will transmit −5 V to the column selector. Each column line, with its transistors, neighbouring ground line, and load transistor effectively acts as a multi-input NOR gate.

Figure 5 also shows how this array of cells can be built on an IC chip with a very high integration density (thereby giving a very low cost per stored bit). Each row line is simply a long metal strip over the oxide layer; each column line and ground line is a long p-region that runs crossways under all the row lines.

When a transistor has to permanently store a 1, the oxide layer is made to be very thin under the metal where it crosses over the silicon between a column line and a ground line. Where the oxide is thicker, on the other hand, the electric charge on the metal is too far from the silicon to switch on a channel, thus making an incomplete transistor that stores a 0.

Besides its high density, the other useful feature of this ROM construction is that the thin oxide areas are created in only one stage of photomasking and etching. This means that manufacturers can **mask-program** customer's *specified information* onto ROM ICs, as easily as they produce standard models.

5. Four 1-bit memory cells: if a 1 is stored each cell is a MOS transistor; if a 0 is stored, each cell is made from an incomplete transistor.



PROMs and EPROMs

Although we said previously that ROMs were preprogrammed during manufacture, there are some types which can be programmed by the user – these are known as **PROMs** (programmable read only direct access memories). This user programming involves the connection of the IC to a special electronic system. The basic PROM IC consists of a matrix of transistors like those discussed earlier. The difference now, is that there is a 'fuse' in series with each transistor. The user can use the PROM programming system to burn out specific fuses, leaving the connected transistors (which represent 1s) in the locations that make up the desired words.

Another type of programmable ROM is also erasable by the user and is known as an **EPROM** (erasable, programmable read only direct access memory). The ability to do this allows considerable flexibility in circuit design, as the existing **firmware** (programmable hardware) can be updated

instead of being replaced.

EPROMs use **floating gate** MOS transistors. This means that the gate is electrically insulated from the rest of the circuit under normal conditions. However, it is possible to set up a negative charge on these gates, which generates a conduction channel in the transistor, between the source and drain. EPROMs are erased by exposing them to strong ultraviolet light, which disperses the electrical charge accumulated on the gates.

Electrically erasable PROMs (**E^{2}PROMs** or **EEPROMs**) and electrically alterable ROMs (**EAROMs**) are recent developments in direct memory technology. Both of these types of device utilise the same floating gate technology used in 'ordinary' erasable PROMs, but employ direct electrical methods to disperse the collected charges.

E^{2}PROMs are erased by a large electrical signal that is applied to the whole circuit, which is then reprogrammed in same way as an EPROM.

EAROMs, on the other hand, could be more accurately described as *read mainly memories*. These devices can be written onto, but the writing process takes much more time than reading. While these devices cannot be used as conventional RAMs, they have the advantage that they can be selectively reprogrammed *in situ*.

RAM

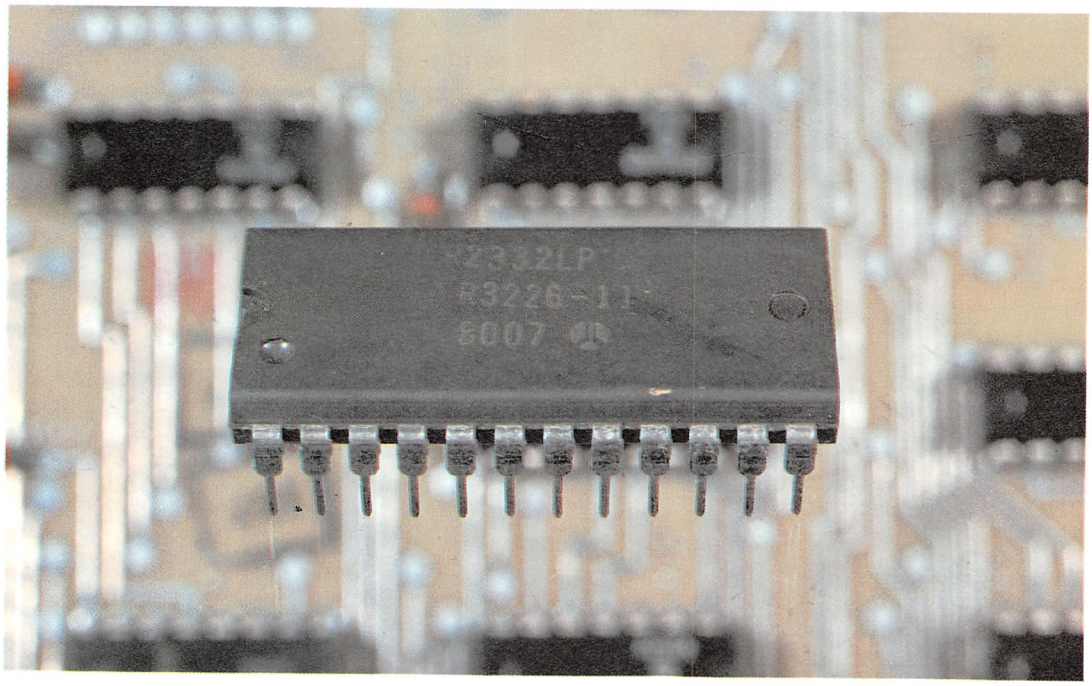
Although the term RAM is an acronym for random access memory, it is more accurately described as read/write direct access memory.

As with shift registers, there are also two general types of RAM: **static** and **dynamic**. Dynamic RAMs store bits in the form of electrical charges, while static RAMs use flip-flops.

How dynamic RAM works

Dynamic RAMs fit into the same general pattern of direct access memories that we saw in *figure 2*. Electrical charges are stored in and read from memory cells through the column lines, using the appropriate switching circuitry in the column selector section. As with all dynamic storage units, the stored charge decays in a fraction of a second, necessitating periodic refresh.

Figure 6 illustrates the general layout of a dynamic RAM. Although only four memory cells are shown, one integrated circuit of surface area 4 mm^2 can store, for example, 16,384 bits in 128 rows and 128 columns. Each specific memory cell consists of one n-channel MOS transistor and a tiny capacitor; each capacitor is made simply from a small area of metal fixed over the chip's oxide layer. When a



Neville Miles

Left: PROM showing connecting pins.

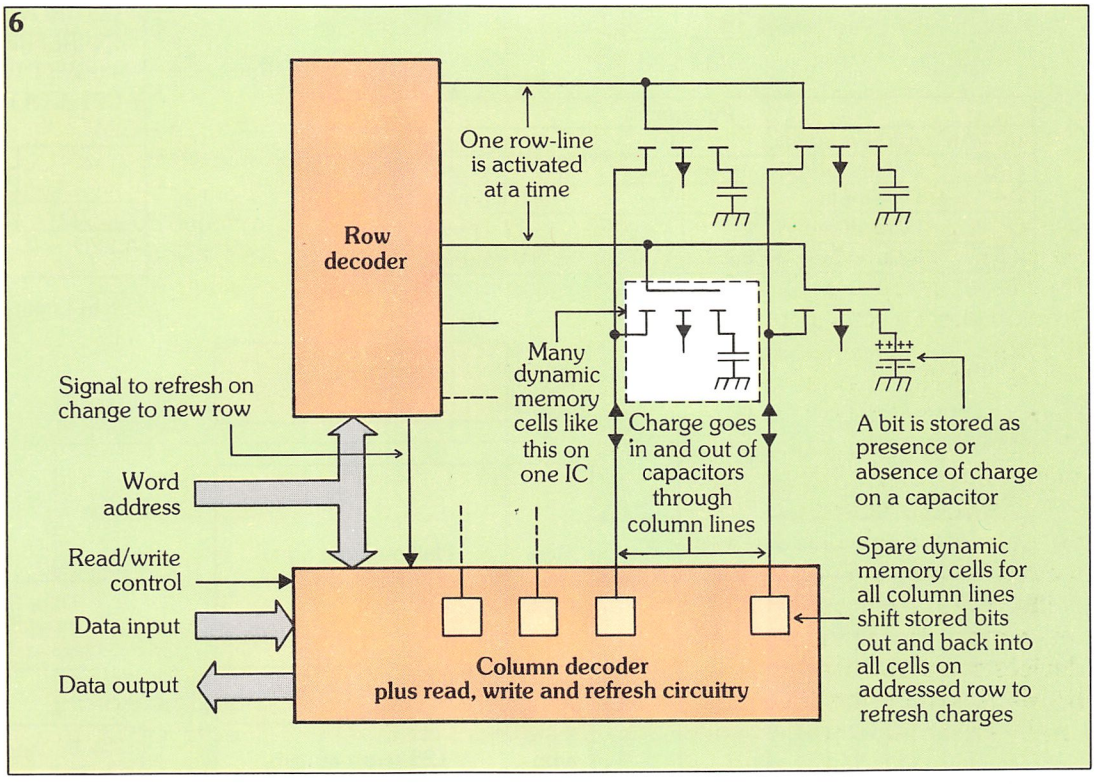
row-line is activated (with a higher voltage), all the n-channel transistors on that row are turned on, connecting their capacitors to the column lines. The column lines charge the capacitors when writing and these charges are detected when reading.

We will not go into further detail for the moment – it is sufficient for our purposes to understand that the words coming from the data input in *figure 6* are written as electrical charges through the column lines, and read out in the same way. Reading and writing operations are handled by the subsystem's read/write control. A memory such as this can handle words of any length up to 128 bits (that is why a word length has not been specified in the diagram).

Each time a row-line is activated, the charges in all the cells on that line are automatically refreshed. This is done by an auxiliary dynamic memory cell which is shown at the lower end of each column line. The stored charges are moved from the cells on the accessed row, into the spare memory cells and then back again with renewed strength. This process resembles the shifting of bits in the dynamic shift register discussed in the previous chapter. Any reading or writing that has to be done takes place at the same time, for selected columns. Remember, we noted earlier that the stored charges need to be refreshed to prevent data from being lost because of the inevitable charge leakage.

To make sure that the memory cells are refreshed often enough, the system controller (which is not shown in our diagram) stops work for about 50 μs every 2 ms. The controller addresses one word on each of the 128 rows during this time, and this triggers the automatic refreshing process.

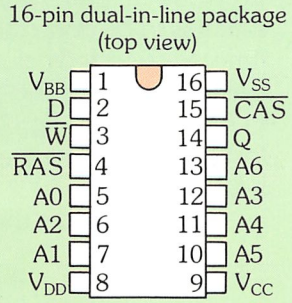
6. General layout of a dynamic RAM.



The 4116 dynamic NMOS 16K RAM

Figure 7 shows the pin layout, logic diagram and functional block diagram for a 16,384-bit dynamic RAM – the 4116. This is a monolithic device, comprising 16,384 RAM cells, organised as 16,384 1-bit words. The device employs single transistor storage cells made using n-channel silicon gate technology. All inputs and outputs of this device are compatible with series 74 TTL circuits. The typical power dissipation is less than 350 mW

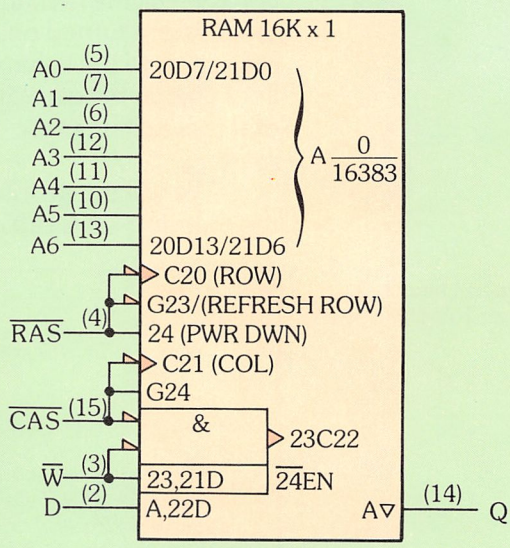
7



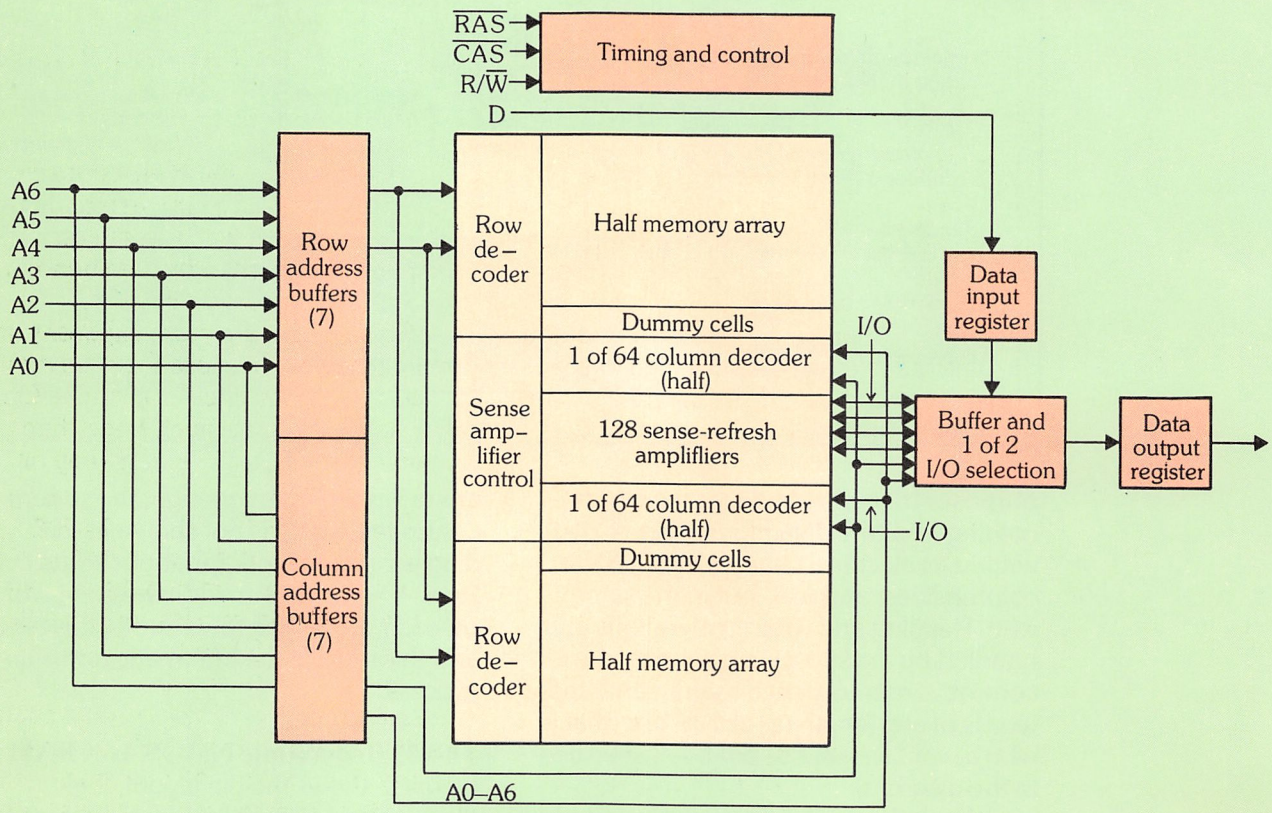
4116 NL

Pin names			
A0 – A6	Address inputs	$\overline{W}$	Write enable
$\overline{CAS}$	Column address strobe	V_{BB}	–5 V power supply
D	Data input	V_{CC}	+5 V power supply
Q	Data output	V_{DD}	+12 V power supply
$\overline{RAS}$	Row address strobe	V_{SS}	0 V ground

a) Pin layout



b) Logic diagram



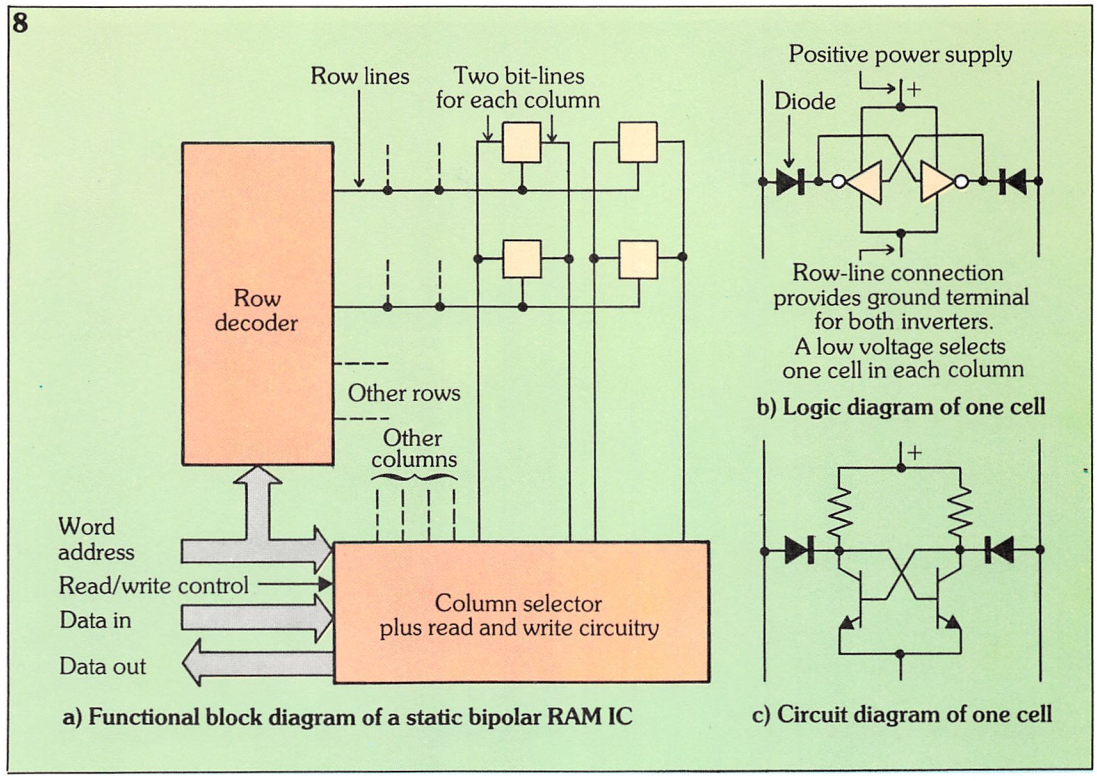
c) Functional block diagram

when the IC is working. Only 10 mW average power is needed to retain data, and this includes the power used to refresh the contents of the memory.

Fourteen address bits are required to decode one of the 16,384 memory cell locations. Seven row address bits are set up on pins A0 to A6 and are latched onto the chip by the row-address strobe ($\overline{RAS}$). Then the seven column address bits are set up on pins A0 to A6 and are latched onto the chip by the column address strobe

7. (a) Pin layout; (b) logic diagram; and **(c)** functional block diagram of a 16,384-bit dynamic RAM – the 4116.

8. Basic arrangement of a static RAM matrix using diode coupled bipolar flip-flop memory cells.



($\overline{CAS}$). $\overline{RAS}$ is similar to a chip enable function as it activates the sense amplifiers as well as the row decoder. $\overline{CAS}$ is used as a chip select pin as it activates the column decoder and the input and output buffers.

The read or write mode is selected by the write enable ($\overline{W}$) input. A high logic signal applied to $\overline{W}$, selects the read mode; a low logic signal selects the write mode.

Data is written onto the IC by selecting the address required, placing a high signal on $\overline{W}$ and sending the data down the D line.

Data is read from the chip by selecting the address required and putting a low signal on $\overline{W}$. The output data is then presented at Q.

The information stored on this chip must be refreshed at least every two milliseconds. This is done by strobing (sequentially switching on) each of the 128 row addresses (A0 to A6) with $\overline{RAS}$.

How static RAM works

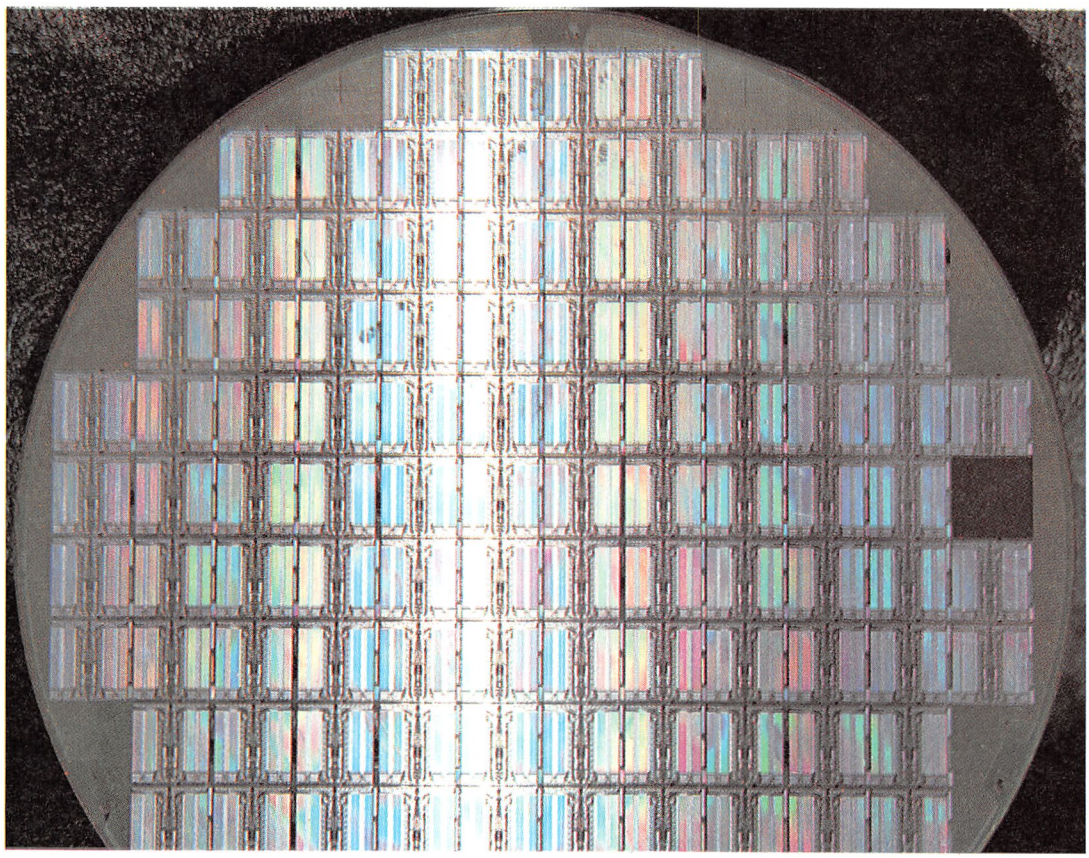
As we now know, static RAM cells differ from dynamic RAMs in that they do not need to be refreshed. Static RAMs use a flip-flop for each memory cell, arranged in a matrix formation. There are many different kinds of static RAM but we can, as a rule, say that they have less complex select and control circuitry than dynamic RAMs. This is because there are no refreshing operations which demand the use of strictly timed data transfer.

Static RAMs are more expensive than dynamic devices because they use more individual semiconductor elements. On the other hand, they are much faster than dynamic devices, as they don't rely on capacitor storage.

Static RAMs can be made using bipolar or MOS transistor technology: bipolar devices are faster, but each storage cell is less compact and less energy efficient than those of MOS RAMs.

An example of the basic arrangement of a static RAM matrix using diode coupled bipolar flip-flop memory cells is shown in *figure 8*. Each cell has two column connections known as bit-lines. As you can see in the logic diagram (*figure 8b*), each bit-line is connected via a diode to the output of one inverter and the input of the other. The diodes act as one-way valves and prevent the cells in one column from interacting with each other.

The row-line connection to each cell provides a ground connection for each inverter as shown. *Figure 8c* shows just



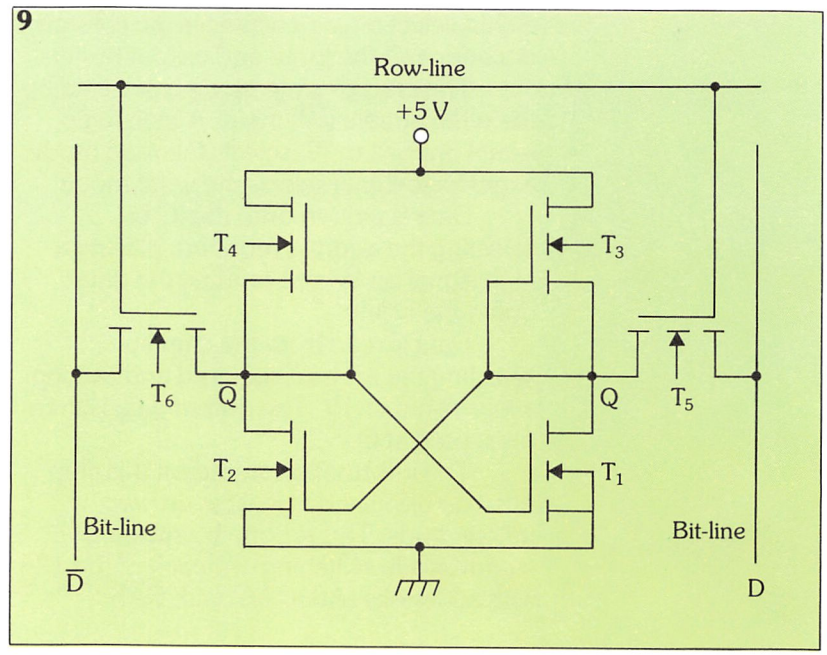
Left: semiconductor wafer containing 64K RAM chips used in the central memory of an IBM 4300. (Photo: IBM)

9. Circuit diagram for a MOS memory cell.

10. (a) Pin layout; (b) logic diagram; and **(c)** functional block diagram for the 2167 static RAM.

how simple the cell's circuit is. If you recall the chapter on flip-flops, then you'll already have identified the cross-coupled inverters as an R-S latch. Each inverter holds the other in its present state until a large voltage on one of the bit-lines upsets the balance and flips the circuit to the opposite state. Data is written to or read from a cell by addressing the row which the selected cell is part of, and writing the data on or reading data from the bit-lines.

MOS RAM circuits are manufactured to perform the same logic function as the bipolar circuit we have just looked at, and a circuit diagram for a MOS memory cell is shown in *figure 9*.



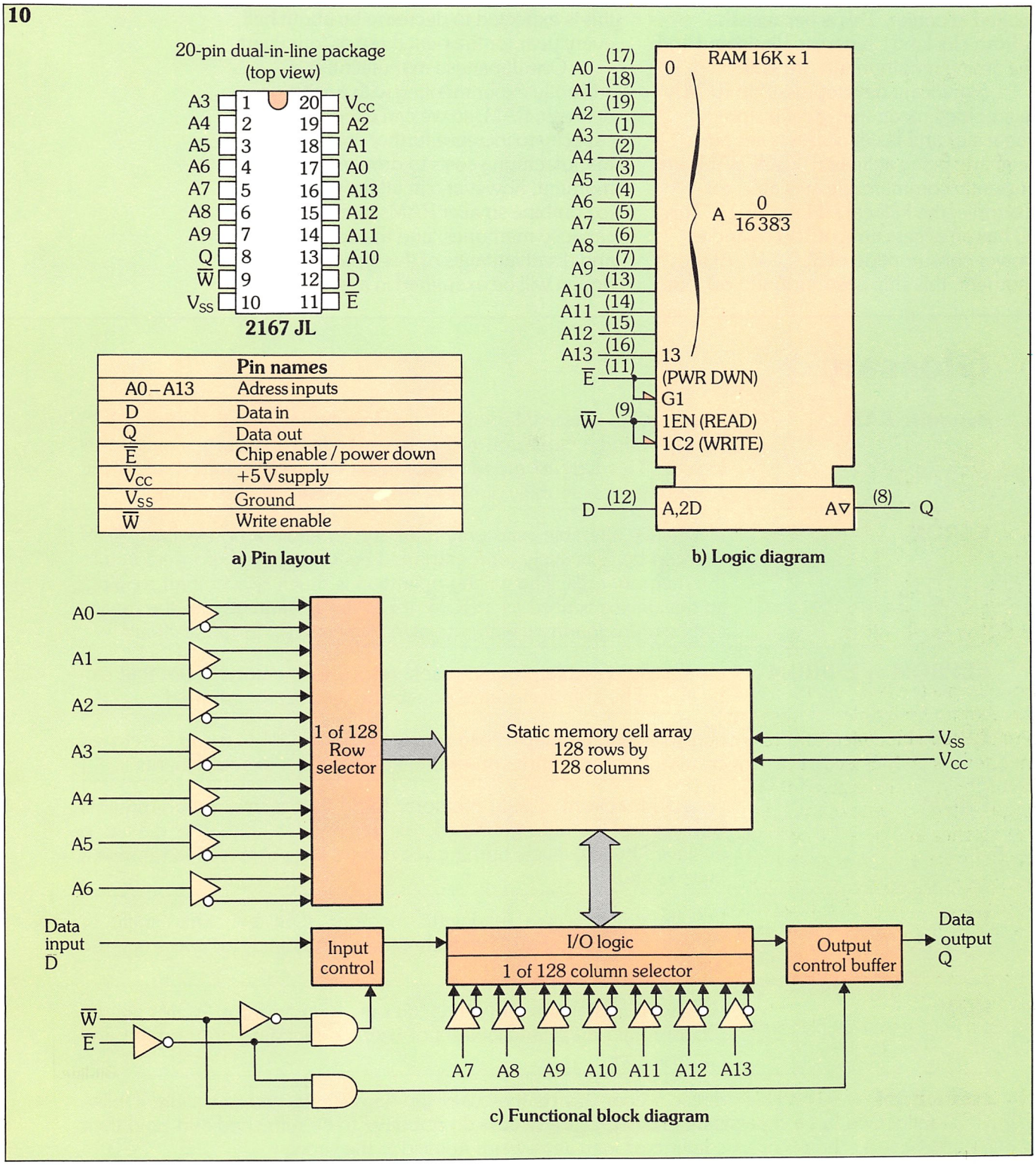
The 2167 static MOS 16K RAM

The 2167 static RAM has a capacity of 16,384 1-bit words. The pin layout, logic symbols and functional block diagram are shown in *figure 10*.

Pins A0 to A13 take the fourteen address bits required to decode one of 16,384 memory locations. A high logic signal applied to $\overline{W}$ selects the read mode, while a low signal selects write. The $\overline{W}$ input can be driven directly from standard TTL circuits.

The data in (D) and data out (Q) terminals are self explanatory, while the chip-enable/power down ($\overline{E}$) connection affects the data in and data out terminals

10



and the internal functioning of the chip. When the chip-enable/power down ($\overline{E}$) terminal is low, the device is operational, i.e. the input and output terminals are enabled and data can be read or written. When this terminal is high, the IC is cut off from the external circuit and is put into a reduced power standby mode. This retains the data stored but I/O operations cannot be performed.

The overhead costs of this circuit are much lower than those of say, the 4116 we

looked at earlier. This is because the refresh clocking circuitry is eliminated and the timing requirements are simplified.

Significant developments in RAM technology mean that greater storage capacities and faster access times are regularly being achieved: 256K RAMs are currently commercially available, for example, the Hitachi HM50256-15. This IC has an access time of 150 ns and a power consumption of 350 mW. At the moment, this chip costs around £50, but this is expected to decrease by about half, every year for the next three or four years.

One Japanese manufacturer is currently experimenting with a 102K dynamic RAM, so we can expect storage capacity to increase further and for manufacturing costs to drop. At the moment, however, circuit designers have to combine smaller RAMs to make large capacity memories, and the advantages and disadvantages of this type of circuit design will be examined in a later chapter.

Glossary

dynamic RAM	direct access device. Memory cells are made using an n-channel MOS transistor and a capacitor; a charge stored in the capacitor represents logic 1. This type of memory has to be continually refreshed as the stored charges naturally diminish, and are destroyed by reading
EAROM	electrically alterable read only memory. In effect a non-volatile RAM that can be selectively reprogrammed by writing new information into the memory cells. The writing operation is much slower than reading, so this device is used as a ROM. It has the advantage that the device can be reprogrammed without having to remove it from the circuit
EEPROM or E^2PROM	electrically erasable programmable read only memory. ROM that can be erased by a large electrical signal, and then reprogrammed
EPROM	erasable programmable read only memory. ROM that can be erased by ultraviolet light and reprogrammed to the user's requirements
PROM	programmable read only memory. ROM that is user programmable but cannot be reprogrammed, as writing data into the device is achieved by selectively burning out minute fuses in the IC to represent logic 1s and 0s
RAM	volatile direct access read/write memory. Can be either static or dynamic according to the technology employed to construct the device
ROM	directly accessible memory that can only be read from. ROMs can be programmed by manufacturers or users, depending on the type and construction
static RAM	direct access device that uses flip-flops as its memory cells. Unlike dynamic RAMs, these cells do not have to be refreshed and hold their contents as long as power is applied to the circuit

20

SOLID STATE ELECTRONICS

Introducing optoelectronics

What are optoelectronic devices?

In this and the following few chapters we shall be looking at some of the electronic components which are classified as being **optoelectronic**, i.e. light and electricity interact in some way within them.

These components fall into two important categories:

1) light **sensors**, or **detectors**, which convert light into electricity;

2) light **sources**, or **emitters**, which convert electricity into light.

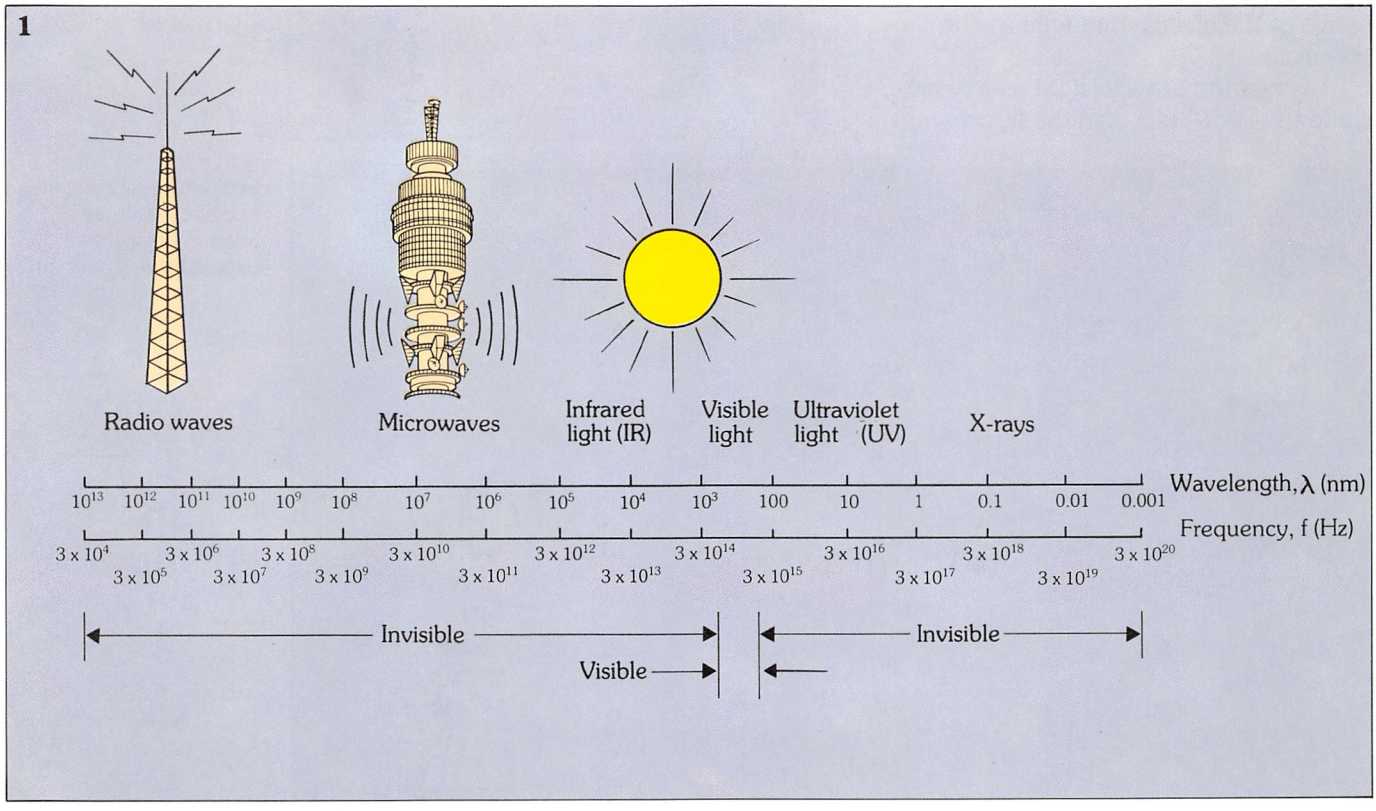
However, before we can examine these components in any depth it is necessary to look at the nature of light itself: where it comes from, how it is transmitted and the terms and units whereby it is described.

What is light?

Light is **radiant energy** and travels in the form of waves, radiating in all directions away from the point of origination. Radiant energy is transferred from the source to the sensor without physical contact. For example, when you stand in front of an electric bar heater (the source), the radiant heat comes to you (the sensor) directly without heating the air between you and the electric bar. The term used to describe such energy transfer is **radiation**.

Radiant energy is the only form of energy that can exist in the absence of matter. One type of radiant energy is known as **electromagnetic radiation** – this comprises the waves of energy associated with electric and magnetic fields. The fields require no supporting medium and can be propogated through space.

1. The electromagnetic spectrum.



Electromagnetic waves travel through space at a constant velocity of approximately 3×10^8 ms^{-1} (186,000 miles per second). The nature of these waves depends on their frequency and this range of frequencies is termed the **electromagnetic spectrum**. The lowest frequencies are radio waves, increasing through microwave, infrared, visible light, ultraviolet, X-rays and gamma rays (*figure 1*). What we know as visible light forms only a tiny part of the whole spectrum.

Origins of light

Electromagnetic radiation is emitted from atoms, when they are excited by heat, chemical action, or other means, in packets of energy called **photons**.

There has been a long scientific argument as to whether photons behaved as waves or particles; recent theories suggest that light exhibits the properties of both.

Natural and artificial light sources

The sun is our most important natural source of light. Sunlight is emitted from the extreme atomic reactions in the sun. The sun produces radiant energy not only in the visible spectrum but also across other bands of the electromagnetic spectrum.

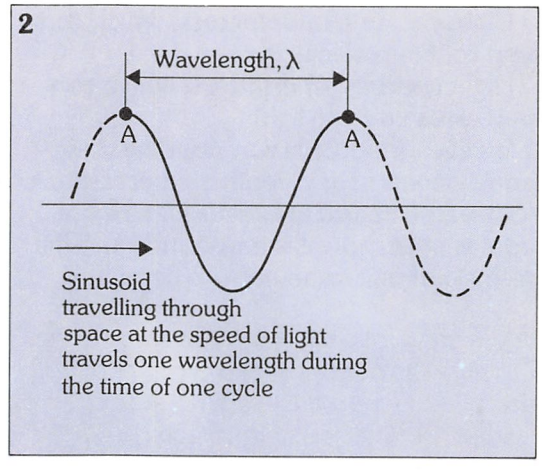
Common artificial light sources include the light-bulb, and the fluorescent tube; such sources are **polychromatic**, radiating light of a number of frequencies. The combination of these various frequencies produces the overall 'white' colour which the eye perceives.

A second important artificial light source is the **laser**; here, the light source is **monochromatic**, i.e. all the photons emitted by a laser have identical wavelengths and energy, and so a pure light of a single colour is produced.

Units and terms

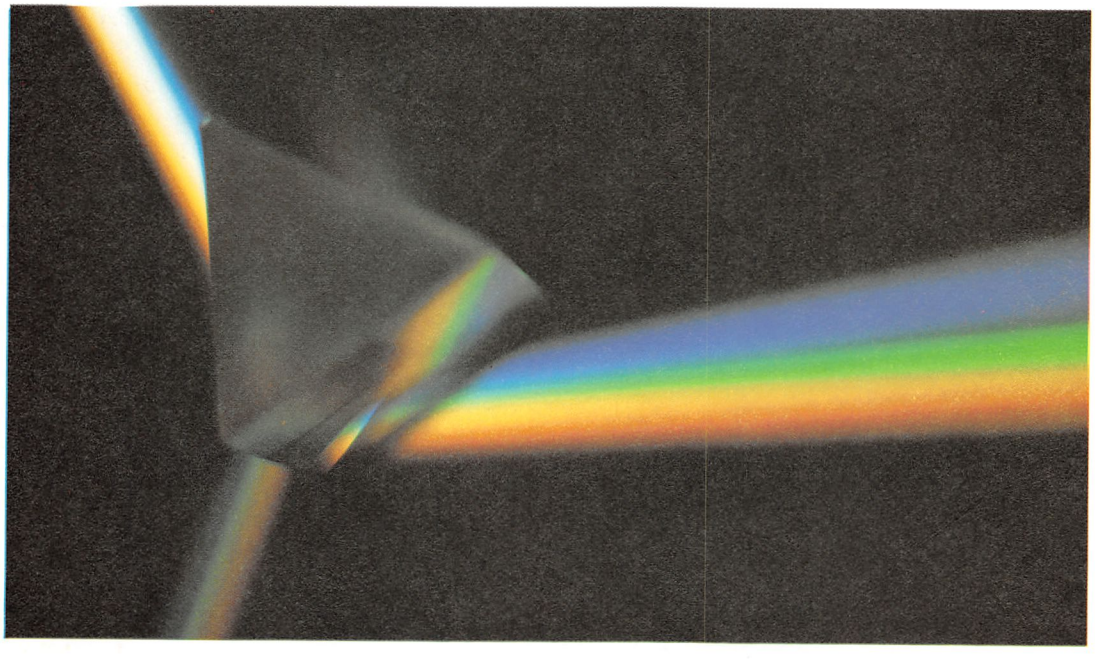
Electromagnetic waves are, in fact, sinusoid curves. The **wavelength**, λ, of these waves is shown as the distance between identical points on two consecutive waves (*figure 2*). (The points at which this measurement is taken are arbitrary.)

Far right: scanning macrophotograph of an LED.



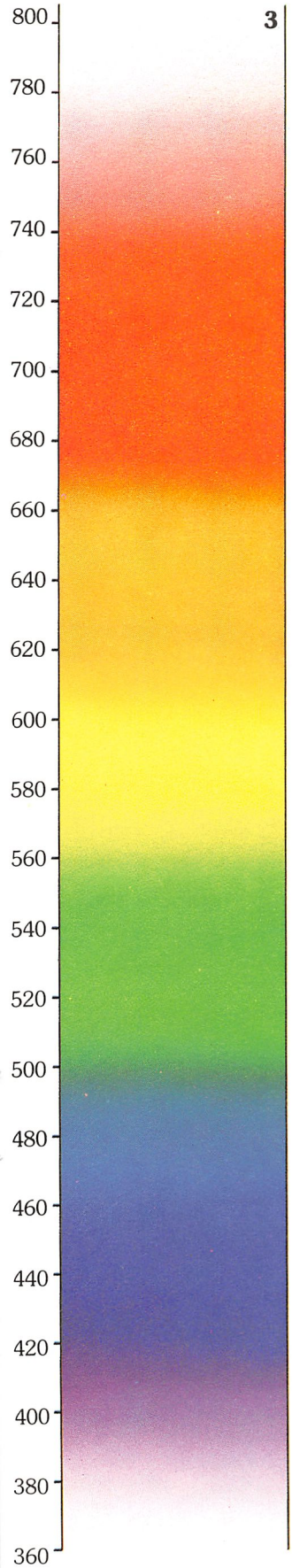
2. The wavelength, λ, is the distance between identical points on two consecutive waves.

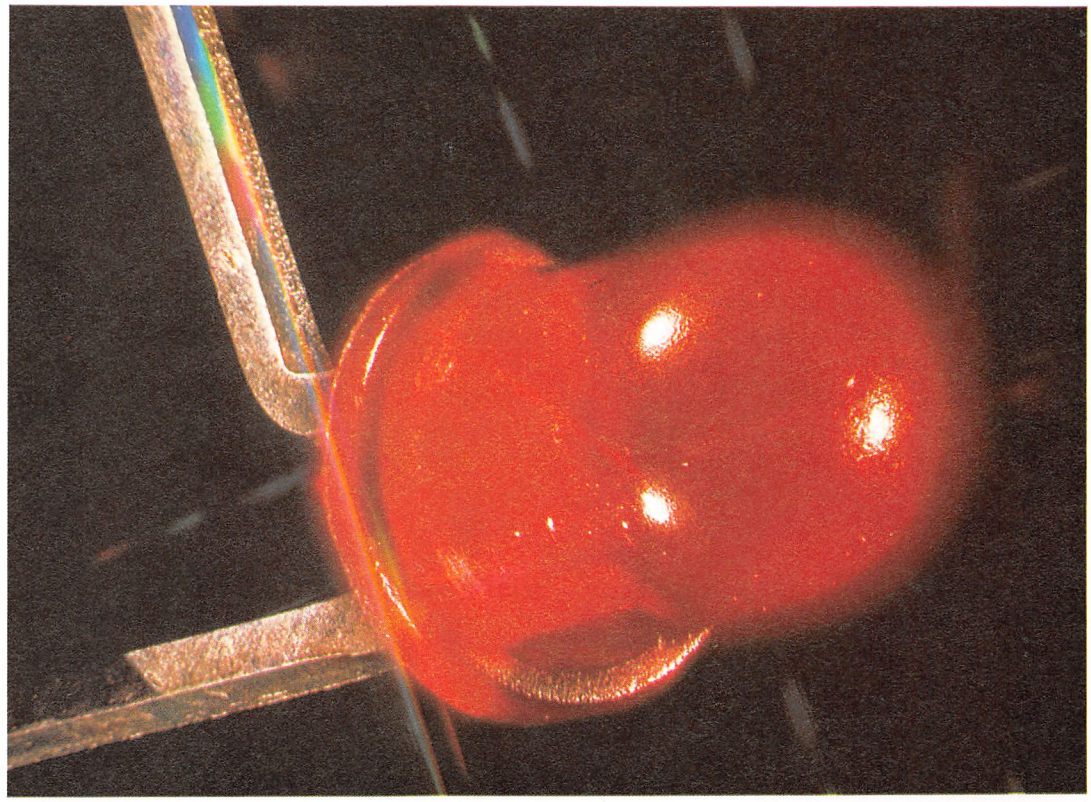
3. The visible spectrum.



Left: white light is split into its constituent wavelengths when passed through a prism.

Paul Brierley





Science Photo Library/Darwin Dale

The **frequency**, f, of a wave is the number of times the wave repeats itself, i.e. **cycles** in a second, and is measured in Hertz, Hz. Although the frequency and the wavelength of each type of electromagnetic wave differ, their speed is constant and the same for all waves. This speed, c, is also constant through *any* **medium** (air, glass, space, etc.).

Electromagnetic wavelength is related to frequency by the expression:

$$c = f\lambda$$

One final term which we need to define is the wave's **period**, T, the time it takes for one cycle to occur. Frequency and period are related by the equation:

$$f = \frac{1}{T}$$

where f is in Hertz and t is in seconds.

We can see that with the high frequencies involved with electromagnetic waves (visible light, for example, has a frequency of about 5×10^{14} Hz) very small wavelengths (about 600 nm for light) and short periods (about 2×10^{-5}s) are associated.

The colour of light

Human eyes respond only to wavelengths between about 400 and 700 nm which, as shown in *figures 1* and *3*, is that portion of the electromagnetic spectrum we call visible light. If the light seen by the eye contains approximately equal amounts of energy for all wavelengths in this range, the eye produces an output that is sensed by the brain as *white light*. If certain wavelengths have more energy than others, the eye's output is sensed as one or more colours.

As shown in *figure 3*, colour is the name given to the response of the human eye to the different wavelengths of light in the visible range. The colour we call red has the longest wavelength at about 700 nm; violet has the shortest, about 400 nm.

When white light is refracted through a glass prism, it is broken down into its constituent colours – progressing from red to orange and yellow to green and through to blue and violet. The same effect, although not as sharply defined, is seen in a natural rainbow.

Coloured light can also be produced by passing white light through a material such as coloured glass or plastic. This material is called a filter because it stops or absorbs some wavelengths of the white

light while allowing other wavelengths to pass through.

Light in optoelectronics

Light is generally used in optoelectronics to transmit information from the source to the sensor via a transmission medium. Both the source and the transmission medium determine the amount of light received by the sensor, therefore information about both can be transmitted to a sensor.

To take a simple case where the characteristics of the medium do not change, the source can be modulated with the desired information and the sensor designed to detect this information. Since the transmission medium doesn't change, it is known that any information received is from the source. This might be referred to as a *non-interrupted* application and would include such applications as fibre optic links and lighted displays, e.g. warning indicators, digital clock displays and digital displays for radios etc.

Another simple case occurs when the source provides illumination while the medium is modulated (*interrupted*). In this case, the sensor is designed to detect the modulated light while suppressing effects of ambient (surrounding) light conditions. Applications include intrusion alarms systems, optical character readers, bar code readers, shaft encoders, smoke detectors, holograms, television, motion pictures and computer graphics.

Human-made light sources are seldom used to transmit high levels of energy except in the case of some laser applications. With laser sources it is possible to concentrate the energy in such a small area that the power generated is high enough to scribe hard materials, cut metals, or weld.

Glossary

electromagnetic spectrum	the range of frequencies of electromagnetic waves. The spectrum includes electromagnetic waves of frequencies around 5×10^{14} Hz, known as visible light
electromagnetic waves	waves which are associated with electric and magnetic fields
frequency, f(Hz)	the number of times a wave repeats itself in a second
radiation	an energy transfer which takes place between two points without the need of physical contact between the two points
wavelength, λ	the distance between identical points on two consecutive waves. Related to the frequency of the wave by the formula: $\lambda = \frac{c}{f}$ where c is the wave's speed
white light	light containing all electromagnetic waves in the visible range at the same intensity. Perceived by the eye as the colour white

21

SOLID STATE ELECTRONICS

Optoelectronic light sources

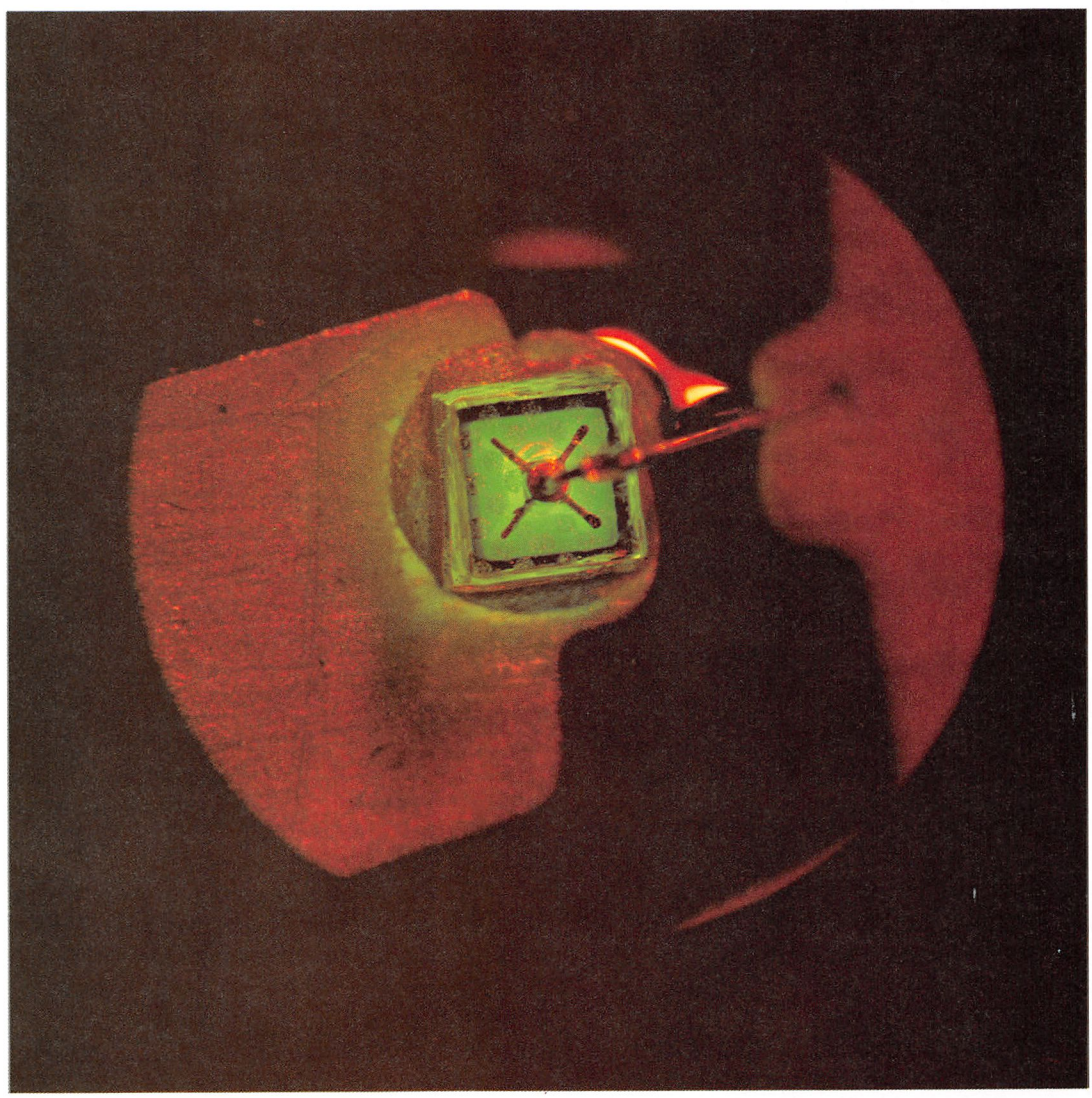
Light emitting diodes

In the introduction to optoelectronics (*Solid State Electronics 20*) we saw that there are two main categories of device: light sources and light sensors. This chapter takes a more detailed look at light sources.

One of the most common optoelectronic light sources is the **light emitting diode** (LED). An LED is a two terminal semiconductor device comprising a p-n junction which conducts in one direction only. This semiconductor material emits light when the p-n junction is forward biased and a current is flowing through it. LEDs can be manufactured to emit visible or invisible (infrared) light.

Visible LEDs are often used as indicators in electronic equipment, either singly to indicate, say, when equipment is on, or in arrays indicating, say, recording volume level of a tape recorder, or calculator and

Right: electroluminescence of an LED.



Paul Brierley

clock displays.

LEDs are reliable and have a very long life if treated carefully, but only limited colours are available. Current consumption (typically about 5 to 20 mA) generally limits the usefulness of an LED to equipment which is not battery powered.

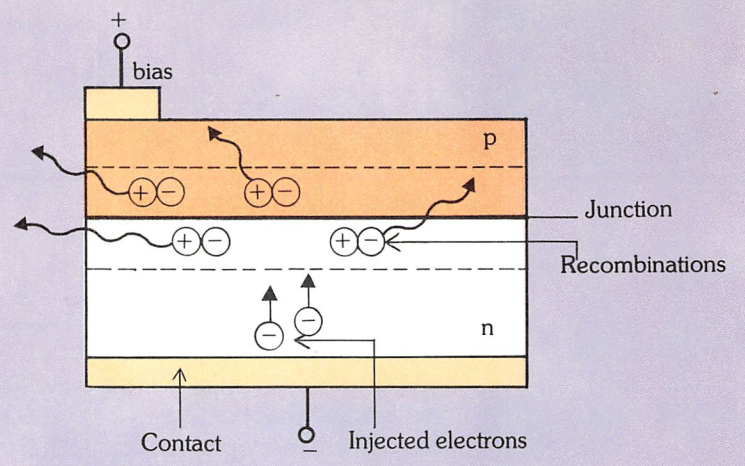
LED operation

The phenomenon which results in the emission of light from a LED is called **electroluminescence** or **injection luminescence**, and is due to the hole/electron recombinations that take place near a forward biased p-n junction.

When electrons are injected into the n-region of a p-n diode, as shown in *figure 1*, and are swept through the region near the junction they recombine with holes in the region. This generates electromagnetic waves of a frequency determined by the difference in the energy levels of the electron and the hole. In order for this recombination to result in luminescence, there must be a net change in the energy levels, and the photon generated must not be recaptured in the material.

The light output power efficiency of LEDs (light output power divided by electrical input power) is low – less than 1%. Also, as temperature increases, the efficiency decreases due to the increase in non-radiative recombinations. LEDs often decrease in efficiency as they age.

1



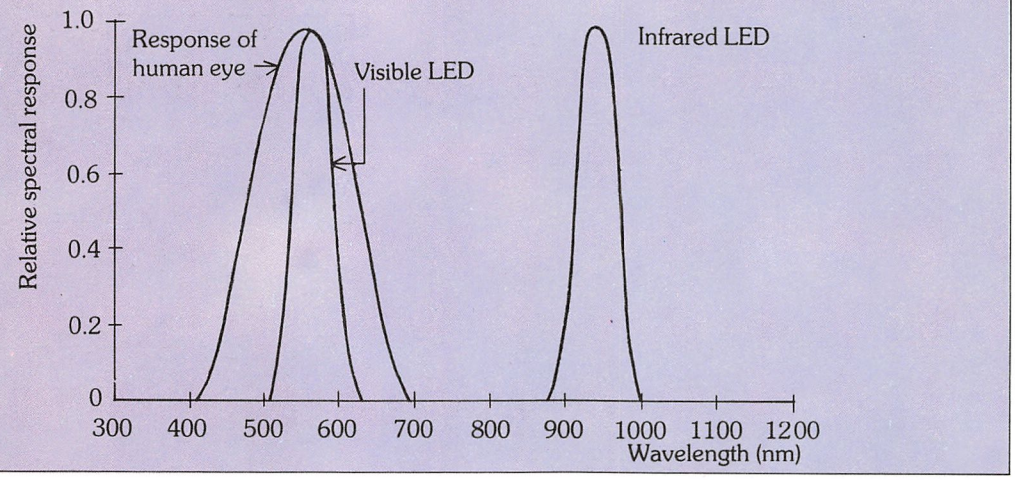
1. The phenomenon of electroluminescence is due to the hole/electron recombinations that occur near a forward-biased p-n junction.

Table 1

Characteristics of a selection of semiconductor materials used to manufacture LEDs

Material	Wavelength	Radiation Range
Gallium antimonide	1770	Infrared
Indium phosphide	985	Infrared
Gallium arsenide	898	Infrared
Gallium arsenide phosphide	650	Red
Gallium phosphide	565	Green
Gallium nitride	400	Violet

2



2. Relative spectral distributions of two LEDs and the human eye.

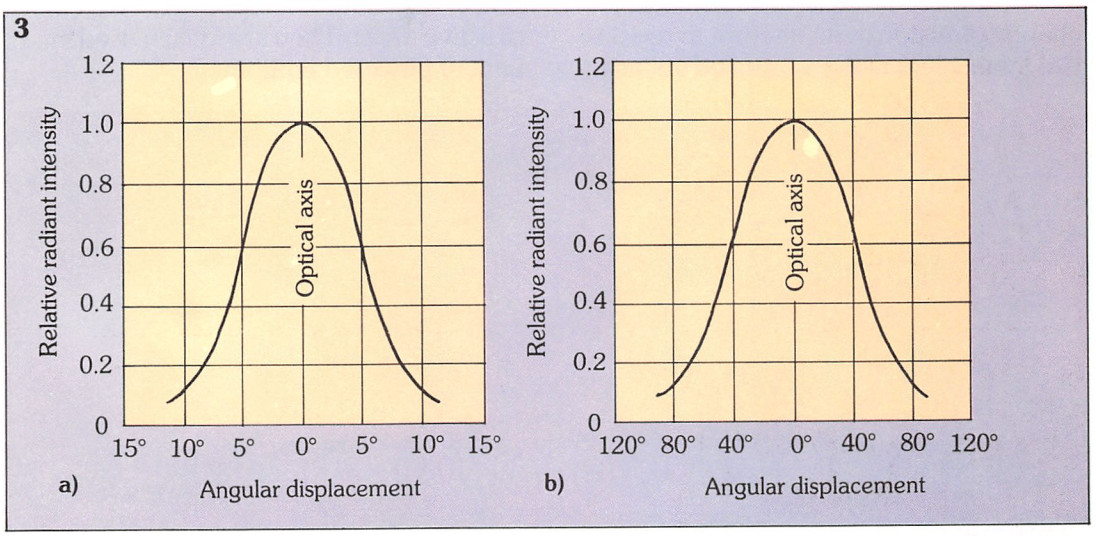
A selection of semiconductor materials used to make LEDs, along with typical wavelengths and radiation ranges of their emitted light, is shown in *table 1*.

Optical characteristics

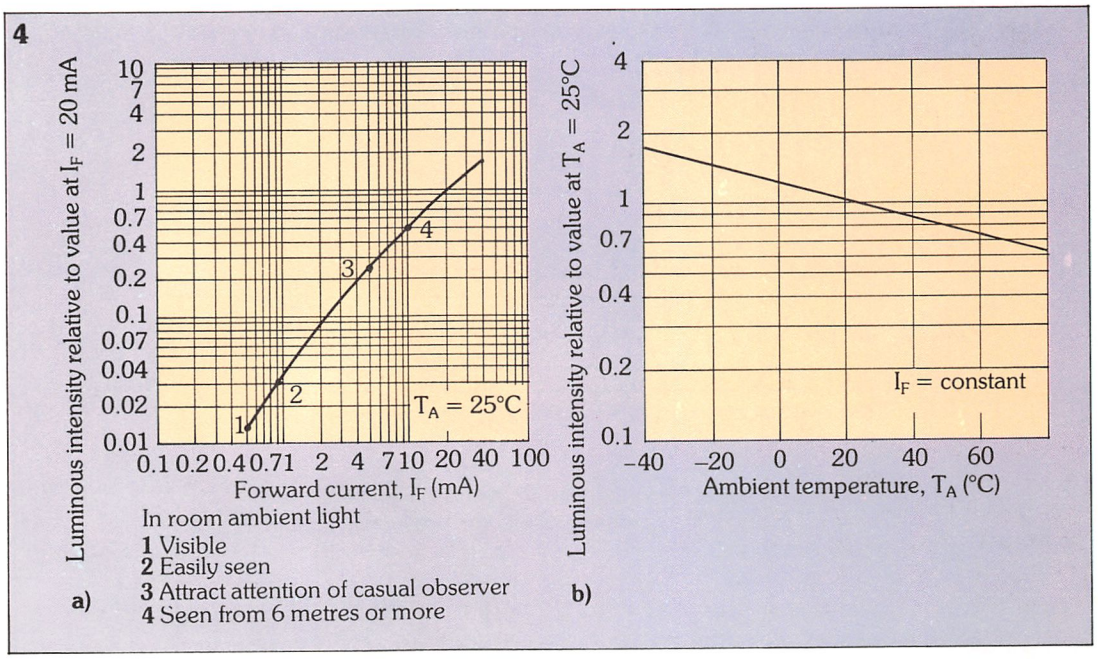
The main optical characteristics of an LED are its spectral distribution and its viewing angle. The **spectral distribution** is the range of light wavelengths which an LED produces and can be displayed as a curve, similar to those shown in *figure 2*. Here, the spectral distributions of two LEDs (one producing visible light, one producing infrared light) are shown, compared with the response of the human eye. We can see that the infrared LED produces no light which is within the eye's response range, and therefore it is impossible to see if the LED is on.

A LED's **viewing angle**, depends more on the actual LED package than on the p-n junction producing the light. Reflectors, diffusers or lenses may be built into the LED package to control the viewing angle. Examples of how viewing angle is affected can be seen in *figure 3* where graphs of relative intensity against angular displacement from the central axis are shown for two types of LED.

Other characteristics of importance are an LED's light output intensity, shown against forward current in *figure 4a*, and against ambient temperature in *figure 4b*.



3. Differences in viewing angle between two LEDs: the TIL31 in **(a)** has a domed lens and a narrow viewing angle; the TIL33 in **(b)** has a flat lens and a wider angle of view.



4. Curves of LED light output against: (a) forward current; and **(b)** ambient temperature.

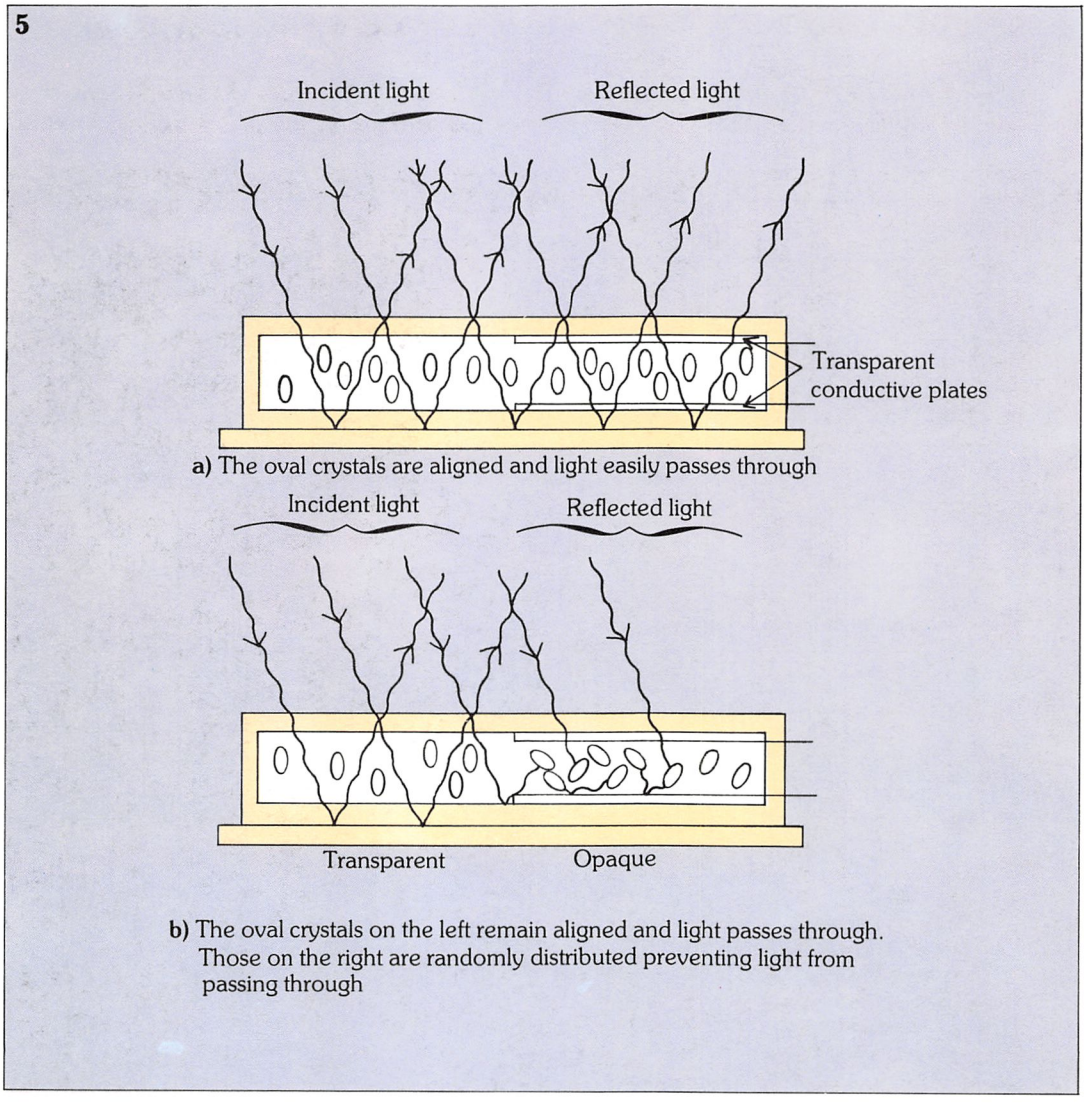
Liquid crystal displays

Liquid crystal displays (LCDs) are not actually light sources – they generate no light, merely filtering incident light, in a controlled manner. The LCDs seen in watches, clocks and hand-held electronic games etc., all work by the same principle.

Two transparent but conductive plates sandwich a layer of liquid crystals which normally all face in the same direction (*figure 5a*). Incident light passes through the liquid crystals of polarised particles fairly easily, and is reflected back through the crystals so that an observer sees a light coloured area.

However, a voltage applied across the plates causes the liquid crystals to change direction in an attempt to repolarise themselves with the applied voltage. As they turn, they interact with the current flowing between the plates and a state of turbulence is created. The moving particles scatter the incident light, randomly reflecting and refracting it – little light is reflected back to the observer, so the area between the transparent plates (*figure 5b*) thus appears dark. Selection of the areas which are turned dark, by using a number of plates and different shaped plates, means that practically any shape of character may be displayed.

Although LCDs exhibit a slow operating speed (a display character can take about 100 ms to change from light to dark) and lack of visibility in the dark, they are very useful because of their extremely low power consumption (about 100 μWcm^{-2} of active area). They are ideally used in battery powered equipment.



5. Operation of an LCD.

Lasers

We have already described the light produced by a laser as being monochromatic i.e. of a single colour and wavelength; it is also **coherent** (in phase) and unidirectional. As a result, the beam of light from a laser does not diverge significantly as it moves through a medium, and thus maintains a high energy density.

To understand how a laser (**l**ight **a**mplification by **s**timulated **e**mission of **r**adiation) works, it helps to think of light as being composed of discrete energy packets, rather than as a wave form. These packets have energy which depends on their frequency of vibration, given by the formula:

$$E = (4.137 \times 10^{-13})\, f$$

when E is the energy in electronvolts (eV) and f is the frequency.

In an atom, electrons orbit the nucleus at specific energy levels. An electron can move from a lower to a higher energy state only if additional energy is provided. For example, if an electron is at an energy level E_1, it can move to higher energy level E_2, only if it receives an additional energy of $E_2 - E_1$ as shown in *figure 6a*. A photon of light with a frequency of:

$$f = \frac{E_2 - E_1}{4.137 \times 10^{-13}}$$

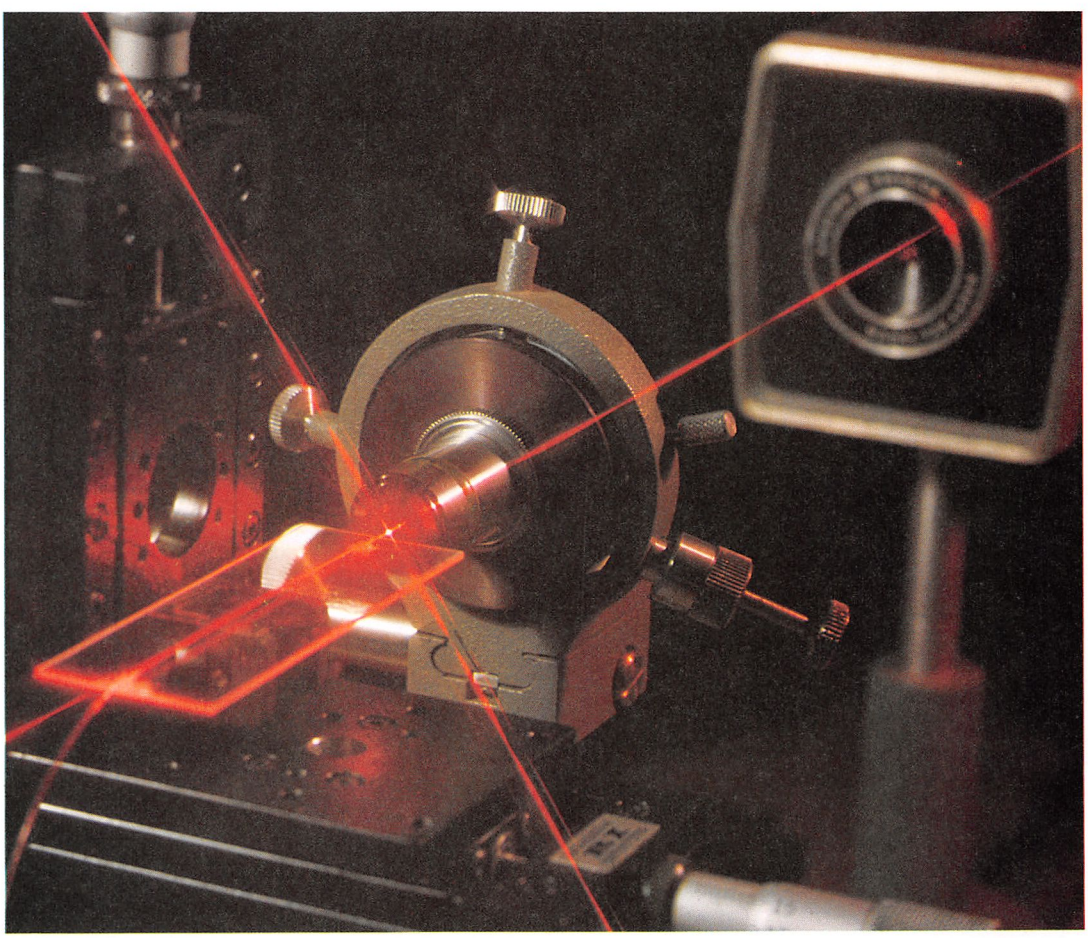
could cause the electron to move to the higher energy level E_2.

When the electron falls back to energy level E_1, as shown in *figure 6b*, it emits a photon of light with the same frequency as the original. This is called **spontaneous emission**.

Stimulated emission occurs if an electron, already in the higher state, is stimulated by a photon of the proper frequency to fall to level E_1: a photon is emitted in phase with the photon that stimulated it, so that one photon in, produces two photons out as shown in *figure 6c*.

The higher energy level is known as the **excited state**; the lower energy level is

Right: ruby laser.



Neville Miles

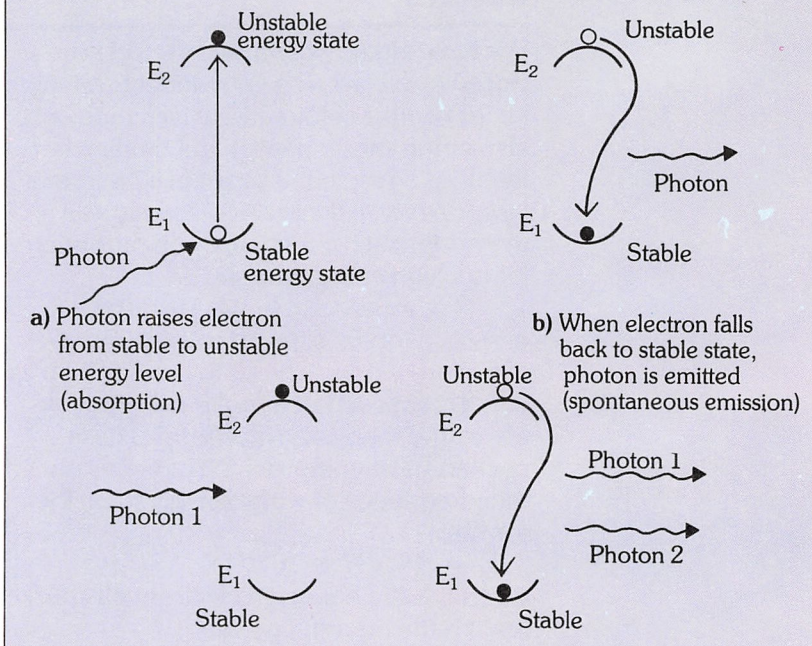
Next page.

6. Photon emission.

called the **ground state**. If a large percentage of the atoms of a material are in the excited state it is said to have a **population inversion**. If a population inversion exists, the probability of stimulated emission of photons improves. The process of creating a population inversion is sometimes called **pumping**: that is, energy from some source must be *pumped* into the laser to provide the additional energy to create the population inversion. Then a photon of the proper frequency creates spontaneous and stimulated emission of two photons; these two photons, in turn, stimulate four photons, and as the process continues, the intensity of the light increases while the phase and wavelength remain the same.

The actual construction of the laser depends on the type of material to be used. Many materials can be used to act as lasers but the most common types are helium-neon, ruby, semiconductor and carbon dioxide.

6



a) Photon raises electron from stable to unstable energy level (absorption)

b) When electron falls back to stable state, photon is emitted (spontaneous emission)

c) Photon 1 stimulates electron that is in unstable level, causing stimulated emission

Glossary

coherent light	light waves which are in phase and monochromatic
electroluminescence, injection luminescence	hole/electron recombinations at a p-n junction which cause an emission of light
infrared LED	an optoelectronic device containing a p-n junction, which emits radiant energy with a wavelength between about 780 nm and 1000 nm when forward biased
laser	light amplification by stimulated emission of radiation
LCD	liquid crystal display. Formed by changing the polarisation of crystals in a layer of liquid, preventing the passage of light
LED	light emitting diode. A semiconductor diode which displays the phenomenon of electroluminescence when forward biased
population inversion	movement of an electron from a ground energy state to an excited energy state so that more atoms are in the excited state than the ground state

ELECTRICAL TECHNOLOGY

Resistance and alternating voltage

As we know, the direct current in a circuit consisting of a pure resistance is related to the applied voltage by Ohm's law:

$$V = IR$$

Now, if we want to find the current flowing through a resistor in response to an alternating voltage (*figure 1*), we can use the same rule, but it must be applied at each separate instance of time. By doing this, we shall discover that if the voltage wave is sinusoidal, the current wave is also sinusoidal and in phase with the voltage wave (*figure 2*). At every moment in time:

$$i = \frac{V}{R}$$

since:

$$v = \hat{V} \sin \omega t$$

then:

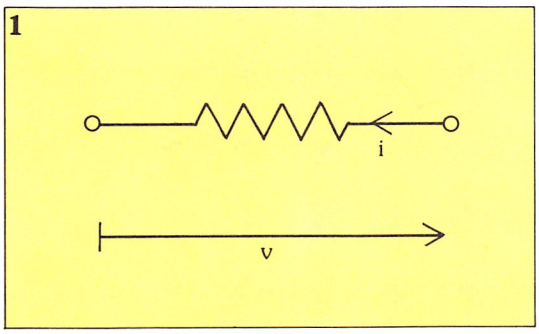
$$i = \frac{\hat{V}}{R} \sin \omega t$$

Notice that the current arrow in *figure 1* points in the direction of conventional current flow, at an arbitrary instance of time. The arrowhead on the voltage arrow indicates the end at which there is a higher potential when the current is flowing in the direction shown.

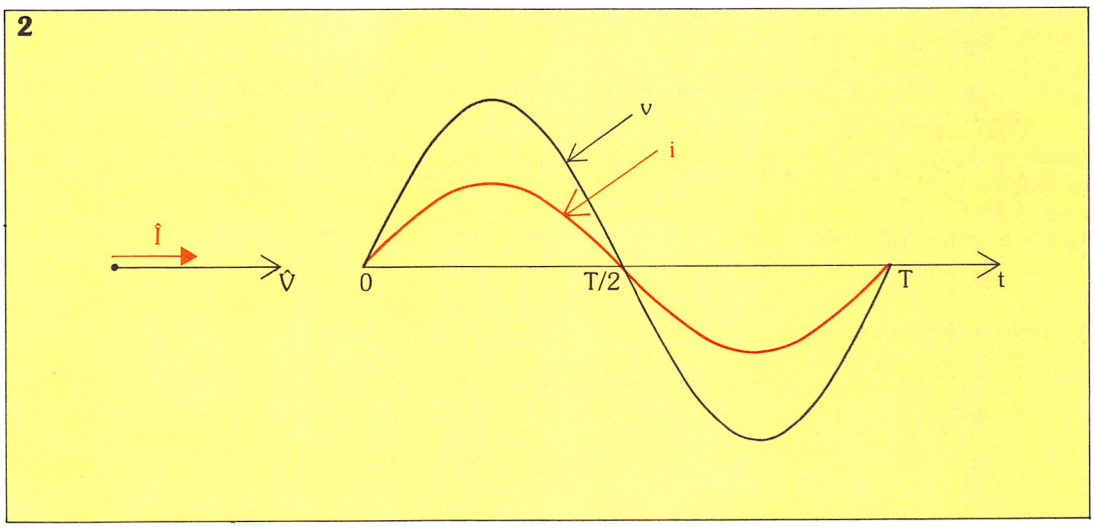
We can now represent v and i by phasors, as is shown to the left of *figure 2*; both phasors are in phase with each other. Here, a different arrowhead is used for current and voltage, as this assists identification in complex phasor diagrams. Although it is possible to solve simple problems like this with a phasor diagram only, we shall draw the sine wave diagram along side.

To find the current that flows through a 7Ω resistor, in response to a voltage of 16 V, a phasor of 16 units length is drawn; the current phasor (in phase with the voltage phasor, because the current is flowing through a resistor) is then also drawn. The current phasor will be of length 16/7 = 2.3 units. The scales of length that are used for current and voltage are only rarely the same, as their numerical magnitudes may differ greatly. It is important to note down these various scales when these problems are being worked out, to save confusion.

1. Determining the current flowing through a resistor in response to an alternating voltage.



2. Sine wave and phasor diagrams for the problem in *figure 1*.



Resistors in parallel

Figure 3a shows two resistors, R_1 and R_2, connected in parallel across a voltage, $\hat{V}$. We can find the total current flowing by using phasor diagrams. The first thing to do is to draw the phasor $\hat{V}$ to scale along the reference

direction. Second, calculate $\hat{I}_1 = \hat{V}/R_1$ and draw a phasor to represent it, to a current scale in phase with $\hat{V}$. Thirdly, draw $\hat{I}_2 = \hat{V}/R_2$, also in phase with $\hat{V}$. Finally, since the current I is the sum of the two branch currents, a phasor for the total current is drawn:

$$\hat{I} = \hat{I}_1 + \hat{I}_2$$

which is also in phase with the supply voltage. *Figure 3b* illustrates the complete phasor diagram. All the lines should be superimposed upon each other, but they are drawn slightly apart for clarity.

If the total resistance is R, then $\hat{I} = \hat{V}/R$ and we can see that:

$$\frac{\hat{V}}{R} = \frac{\hat{V}}{R_1} + \frac{\hat{V}}{R_2}$$

which is the same expression as that obtained when working with continuous (direct) voltage.

Resistors in series

Figure 4a shows two resistors connected in series, and *figure 4b* shows the circuit's phasor diagram. In this case, it is more convenient to draw the current, $\hat{I}$, as our initial phasor along the reference direction, having chosen a suitable scale. This is the most convenient way of developing the phasor diagram, as the same current flows through both the resistors. We can then draw the phasors, $\hat{V}_1 = \hat{I}R_1$ and $\hat{V}_2 = \hat{I}R_2$, to a suitable voltage scale, in phase with $\hat{I}$. Finally, we draw $\hat{V} = \hat{V}_1 + \hat{V}_2$.

If R represents the resistance of the complete circuit, so that $\hat{V} = \hat{I}R$, then we can see that the formula for calculating the total resistance of series resistors in an AC circuit is the same as that for a DC circuit i.e:

$$R = R_1 + R_2$$

Power in resistors with sinusoidal voltages

Let's now look at the power developed in the resistor shown in *figure 1*, when it is connected to a sinusoidal voltage. *Figure 5* shows the voltage and current that are flowing, as functions of time.

The power dissipated at any moment is given by:

$$p = vi = i^2R = \frac{V^2}{R}$$

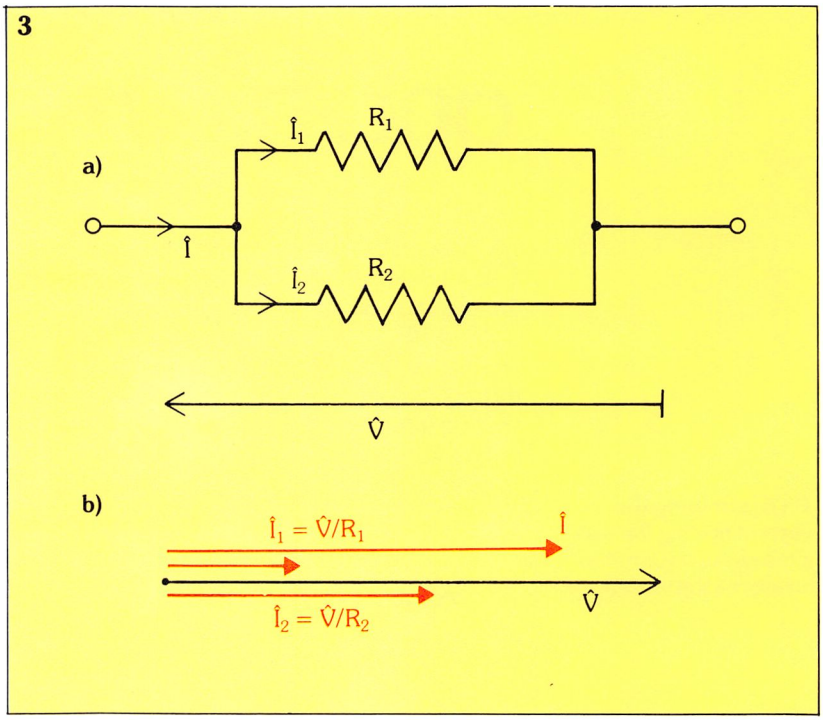
By determining the power at each instant of time using this formula, we shall see that the instantaneous power dissipated is always positive. (Remember, the square of a negative number is always positive.) The power dissipated is indicated by the red curve in *figure 5*, and you'll notice that the power pulsates from zero to maximum and back to zero, twice for each voltage or current cycle. The instantaneous power is zero when the current is zero, and maximum when the current is at either its positive or negative maximum value.

If the current is a sinusoid:

$$i = \hat{I} \sin \omega t$$

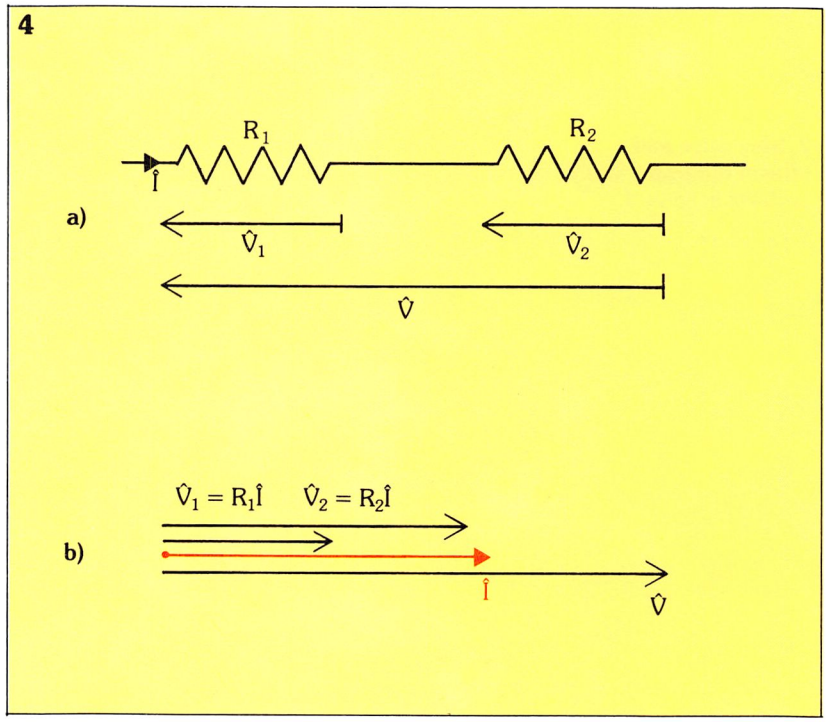
then the maximum value is $\hat{I}$ (usually pronounced I hat) and the maximum value of the power is $\hat{I}^2R$. Over a period of time, T, the total energy (= power x time) given out as heat is equal to the grey areas under the (red) power

3

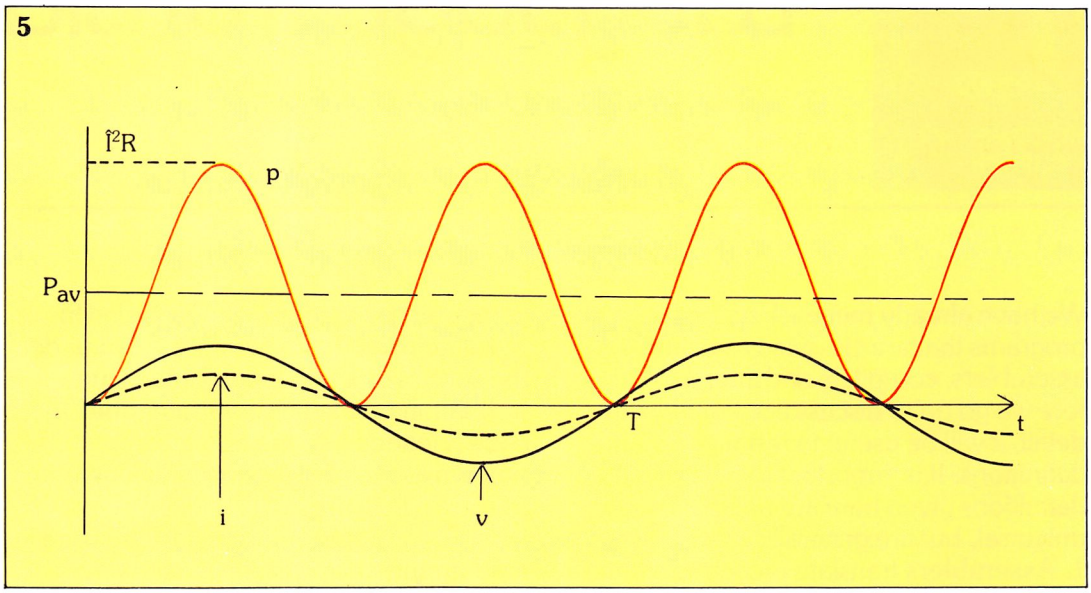


3. (a) Circuit diagram; and **(b)** phasor diagram for two resistors connected in parallel.

4



4. (a) Circuit diagram; and **(b)** phasor diagram for two resistors connected in series.



5. Power developed in the resistor of *figure 1* when connected to a sinusoidal voltage.

curve.

The average power of this period in time is shown by the straight broken line. The average power, P_{av}, has a magnitude equal to half the maximum value:

$$P_{av} = \hat{I}^2 R/2$$

rms power value

As the voltage and current in AC circuits are constantly changing, it is convenient to be able to work out the power dissipated. To do this, we need a figure that will effectively represent the DC equivalents of the sinusoidal voltages and currents for use in power calculations.

If a current, I_{DC}, flows continuously through the resistor R, then the average power dissipated would be:

$$P_{av} = I^2_{DC}R$$

The equivalent direct current is found by equating the last two expressions we have seen for P_{av}, thus:

$$I^2_{DC} = \frac{\hat{I}^2}{2}$$

$$I_{DC} = \frac{\hat{I}}{\sqrt{2}}$$

This value of an alternating current is called the root mean square value, usually abbreviated to **rms** value. This is the value of the direct current that has the same heating effect as the sinusoidal current. Thus the rms current:

$$I = \frac{\hat{I}}{2}$$

$$= 0.707\,\hat{I}$$

where $\hat{I}$ is the maximum value of the sinusoidal current.

The same argument can be used to show that the rms value, V, of a sinusoidal voltage of peak value $\hat{V}$ is:

$$V = 0.707\,\hat{V}$$

Phasor diagrams

So far, when using phasor diagrams, we have always drawn the phasor as being equal to the maximum value of the voltage or current. As sinusoids are usually measured in rms values, we shall in future scale down all phasor diagrams by a factor of 0.707. This means that all phasors will be measured by their rms values, but the shape or configuration of the diagrams will not be affected at all. □

12

BASIC COMPUTER SCIENCE

Language translators

We have already met each of the three programs that are classified as **translators**: assemblers, compilers and interpreters. Before we go on to discuss each of them in detail, it will be useful to restate a few definitions. It is important to note that the definitions given here are not completely universal, but are generally accepted.

1. **Assemblers** translate programs written in assembly code into machine code. One machine instruction is generated for each source instruction and the resulting program can only be executed when the assembly process is complete.
2. **Compilers** translate high-level programming languages into machine code. Many machine instructions are generated for each source instruction and the resulting program can only be executed once the compilation is complete.
3. **Interpreters** check and translate high-level language programs, line-by-line as programs are executed. The generalised translation principle is shown in *figure 1*.
4. The language in which the application program is written is known as the **source code**; the program is known as the **source program** (irrespective of the language in which it is written).
5. **Object code** is the term given to the machine code resulting from the translation; the **object program** is the machine code program.

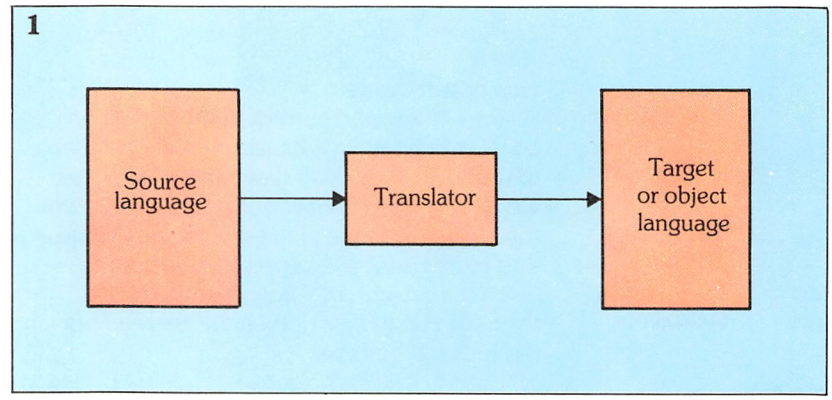
Translators, then, are an aid to human communication with computers as they enable users to write programs in languages more suited to their needs.

Cross translators

There are instances where a translator generates object code for a computer other than the one on which the translator is actually being run. This **cross translation** principle is often used on home computers because they lack the necessary resources to support the translator program. Many computer games cassettes (recorded in machine code) have been recorded under the control of another computer. After translation on the first computer, the object code is loaded into the home computer's (known as the **target computer**) memory as shown in *figure 2*.

The program, stored in ROM, used in a programmable washing machine is another example. Here, the target 'computer' is the microprocessor based washing machine which does not have the memory or interface resources to support a translator program. The washing program is developed on another computer – which may **emulate** the target environment.

1. Generalised principle of translation.

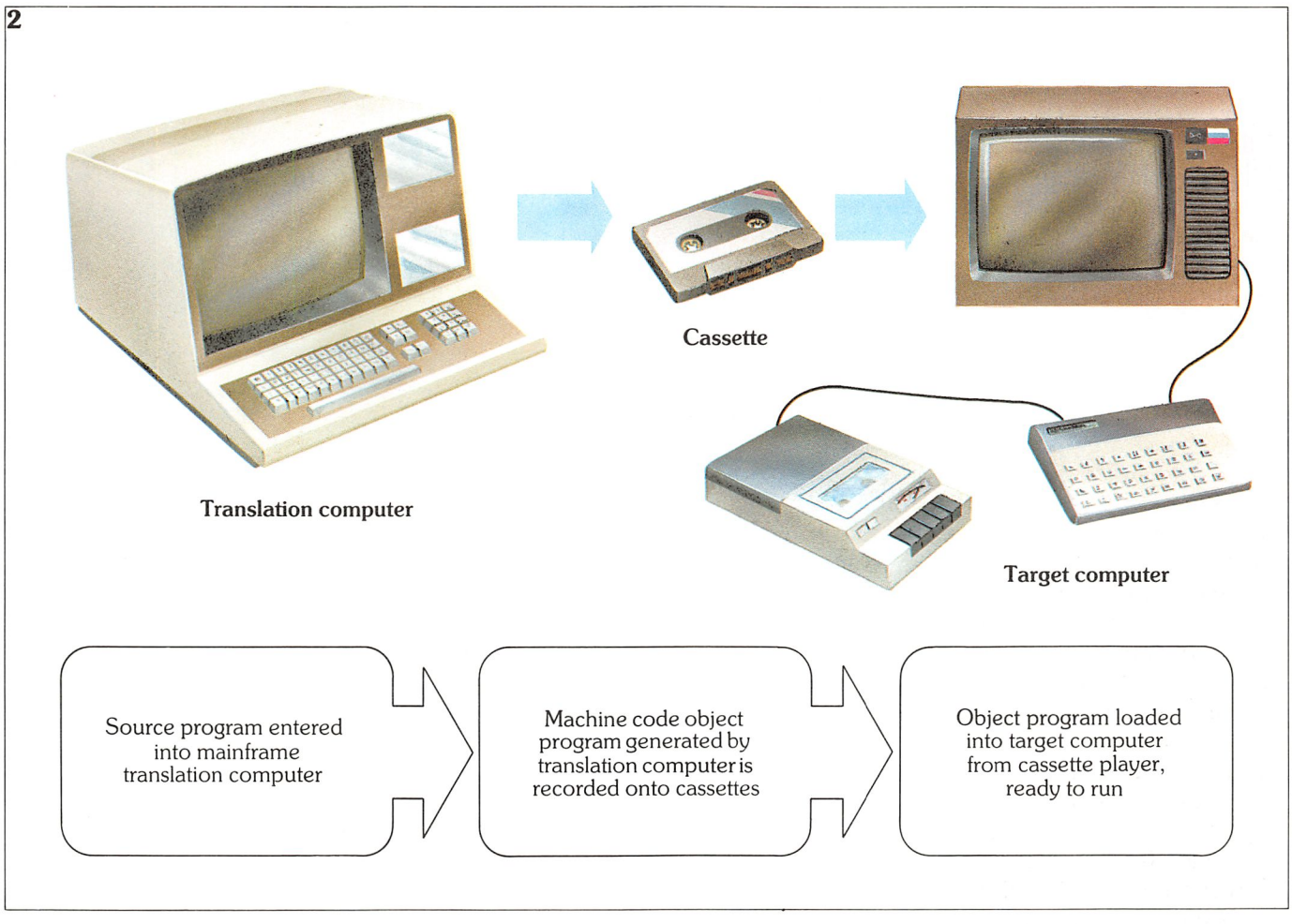


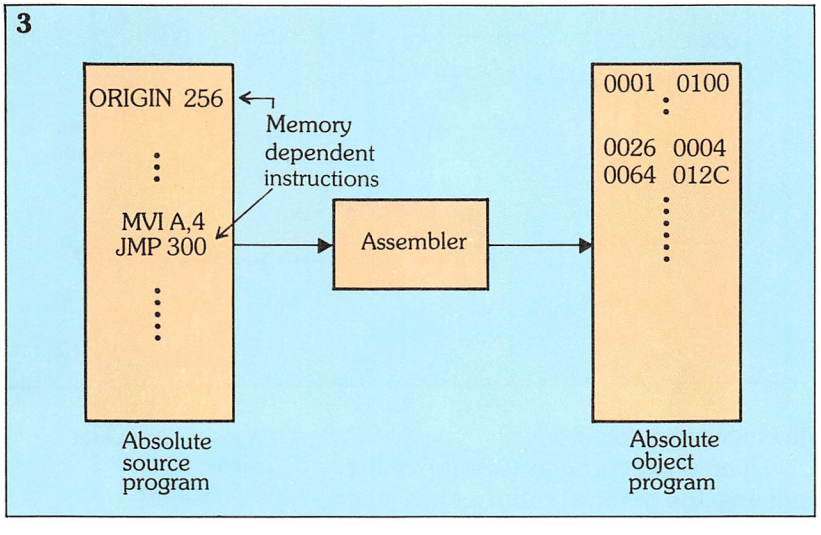
Absolute and relocatable programs

Depending on the type of translator and the computer used, the object program can be said to be either **absolute** or **relocatable**. An absolute object program is one in which the absolute address of each instruction is determined in hardware, and memory locations are specified in **memory dependent instructions**, for example:

ORIGIN 256

This instruction, after translation by the assembler, takes the form of the standard machine coded instruction of an op code, followed by the operand location (both in hex):





2. Cross translation.

3. A memory dependent instruction.

0001 0100

After executing the instruction, the computer then processes the instruction which is held in memory location 256. Obviously, the program containing such a memory dependent instruction will not be correctly executed if memory location 256 does not hold the next correct instruction. These instructions must not, therefore, be moved once the program has been translated and loaded into memory.

A second example of a memory dependent instruction is shown in *figure 3*. The instruction:

JMP 300

causes a transfer of control within the program after processing. The mnemonic JMP is the instruction to jump from the contents of the memory location the computer is processing, to the contents of the memory location specified, i.e. location 300. The machine code instruction (in hex) is: 0064 012C, where 0064 is the op code signifying the jump instruction. The remaining instruction in the object program: MVI A, 4 instructs the computer to store the value 4 in register A. There are no references to memory locations in this instruction and the instruction could be carried out correctly from any memory location.

More commonly, particularly on

mainframes, it is of no importance to the user where the program is actually placed in memory. On multiprogrammed computers, for example, programs are loaded into particular memory locations in accordance with the available space at the time. In such systems, the translator produces **relocatable** object programs in which instructions may occupy different locations each time they are loaded.

To do this, the translator simply produces the object program referenced to start at memory location 0. As it is loaded into memory, the object program's memory references are updated to those where the program will be actually stored. For example, if the program is to be stored in a block of memory starting at location 256, all memory references in the program will have 256 added to them.

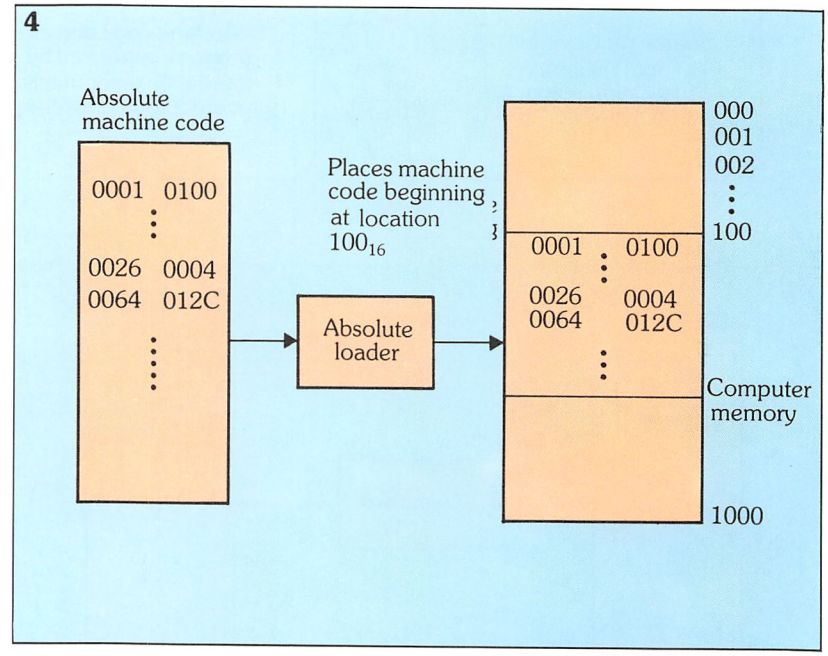
Loading these two different types of object program into computer memory needs to be tackled in different ways. To load an absolute program, for example (*figure 4*), all the **absolute loader** program needs to do is to read the first word from the object program, and then load the instruction into memory at the specified location as memory locations are specified by the translator. Successive object instructions are loaded into successive memory locations, unless other memory referenced instructions occur.

The loader program responsible for loading relocatable object programs, the **linking loader**, has a more difficult job, however. The principle of linking relocatable object programs is shown in *figure 5*. A **relocation directory**, which is a table of all the memory referenced instructions, is generated by the translator and forms part of the total object program.

The linking loader adds the value of the object program's starting memory location to each of the memory referenced instructions, thus relocating the whole program into the block of memory designated by the linking loader. If the object program is to be relocated to the block of memory starting at location 1000 (in hex, as shown in *figure 5*), then the instruction ORIGIN 0 becomes ORIGIN 1000 and the second memory referenced instruction JMP 100 becomes JMP 1100. The non-memory referenced instructions are loaded into successive locations .

If an object program is long, or if it contains one or more sub-routines, it may be divided into smaller, **relocatable modules** – the linking loader then relocates these modules into sections of memory.

A second directory, known as an **external symbol table**, is generated by the translator. This table is used by the linking loader to locate each module of the program when required.



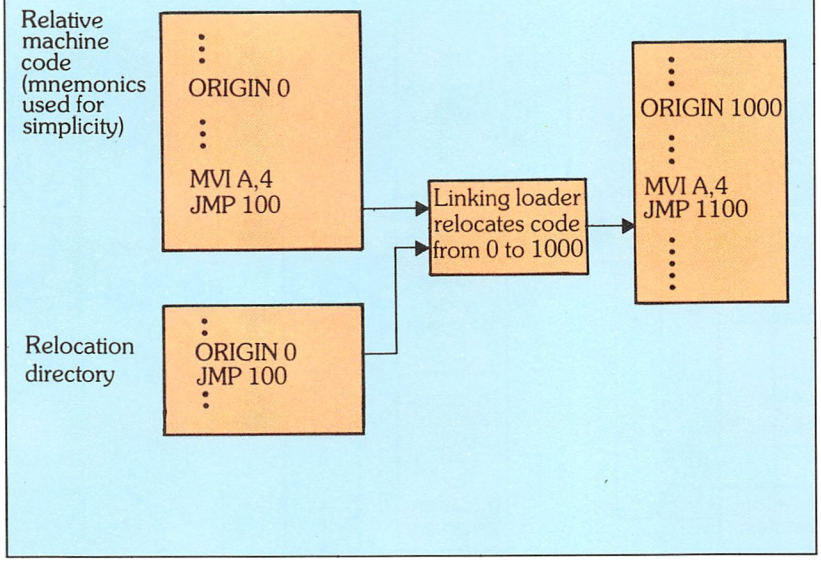
Above: a computer centre. (Photo IBM).

4. Loading absolute machine code.

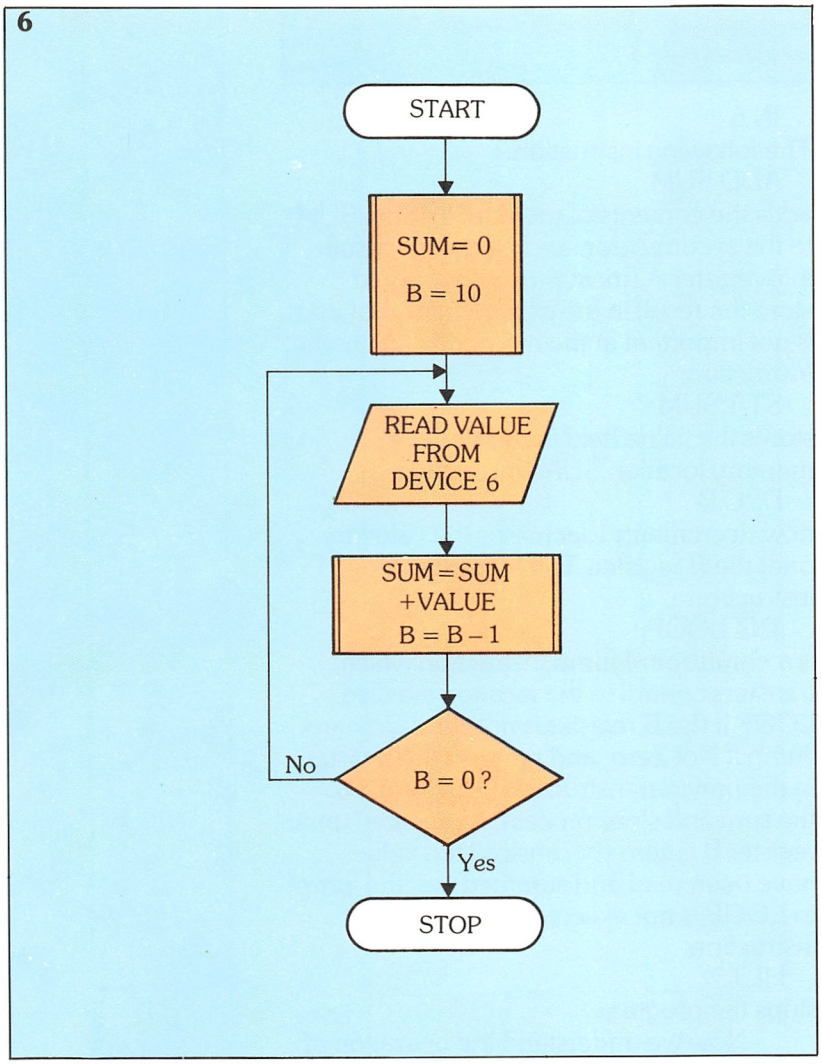
5. Loading relocatable machine code.

6. Flowchart for assembly language program.

5



6



Assemblers

The assembler is responsible for reading the symbolic instruction codes (the mnemonics), operands, and addresses and converting them to machine code. For example, an instruction such as:

MV1 B, 6

must be changed to a machine instruction, say 0026 0006, where 26 is the op code and 6 is the operand to be placed in the B register within the CPU. This instruction can be assembled in **one pass**.

The instruction:

MV1 B, DATA

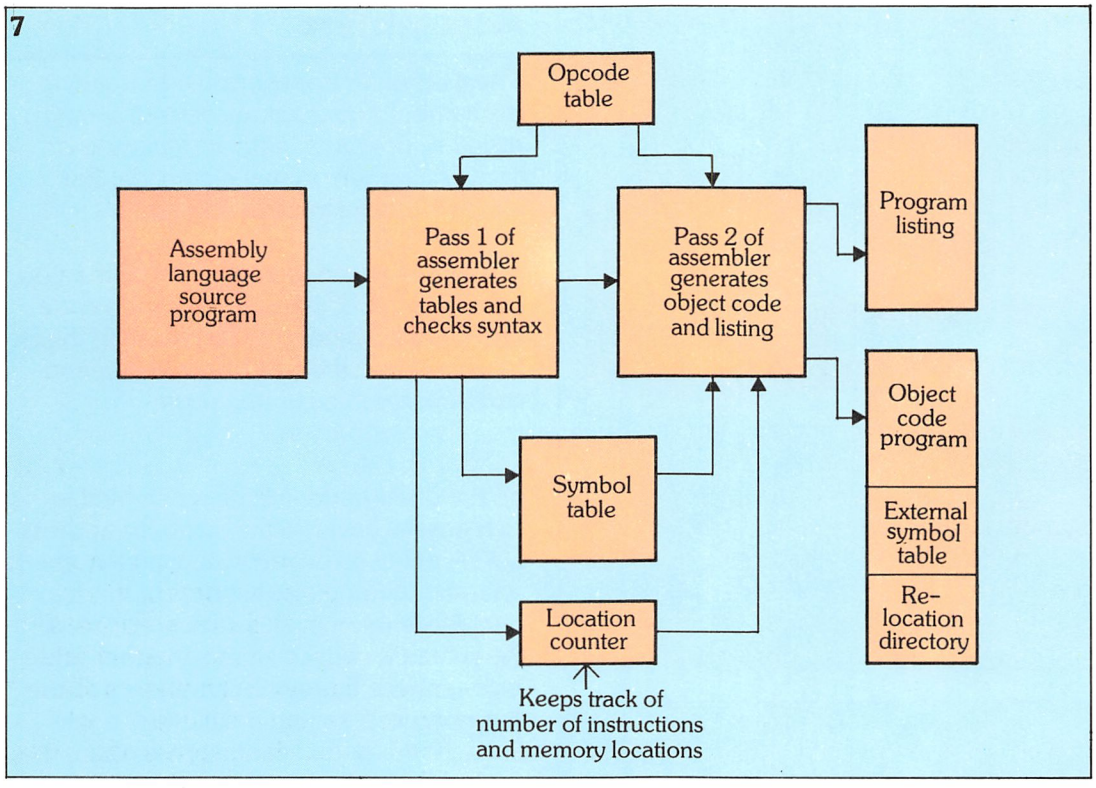
on the other hand, will be assembled in **two passes** because the symbolic address DATA needs to be replaced with a numeric operand. During the first pass of this two pass Assembler, each instruction is read, the syntax is checked and a **symbol table** is generated. The symbol table lists all the symbols used, together with their operands. At the end of the first pass, then, the assembler is in a position to assign appropriate numeric values to these symbols. It also keeps track of the total amount of memory needed in a **memory location counter**.

During the second pass, the assembler inserts the numerical operands giving full translation into machine code object program. On large machines these two passes are handled as one.

As an example, consider a program with the following assembly language instructions:

```
START:  LDA ZERO
        STA SUM
        MVI B, 10
LOOP:   IN 6
        ADD SUM
        STA SUM
        DEC B
        JNZ LOOP
        HLT
ZERO:   DW 0
SUM:    DS 1
        END
```

In this example we are using instructions for two general purpose registers, A and B, and also an accumulator. A flowchart is shown in *figure 6*. This program adds ten numbers together after reading them from input device 6.



7. The two pass assembler process.

The program begins with the instruction:

LDA ZERO

which sets register A to the value of ZERO contained in the symbol table. Looking at the third instruction from the bottom we see that the symbol ZERO is set to the value 0 by the instruction:

ZERO: DW 0

This instruction is known as a **directive** or **pseudo instruction**. It is not a machine instruction, but an instruction *to the assembler* telling it to set the storage location labelled ZERO to the value 0.

SUM DS1

instructs the assembler to label a storage location or memory word with the symbolic name SUM. The contents of this location are then set to zero by the instruction:

STA SUM

i.e. store the value now in register A, in memory location SUM.

Register B is set to the value 10, the number of values which are to be read and summed, by the instruction:

MVI B, 10

which means move the value 10 into register B. Values are read from input device 6 into the accumulator by the instruction:

IN 6

The following instruction:

ADD SUM

adds the contents of memory locaton SUM to the accumulator, and places the result into register A (most processors would store the result in the accumulator, but that is not important at the moment). Again, the instruction:

STA SUM

stores the value held in register A, in memory location SUM. Instruction:

DEC B

now decrements (decreases the value by one) the B register. The following instruction:

JNZ LOOP

is a **conditional jump** instruction which transfers control to the location labelled LOOP if the B register is not zero; it means Jump if Not Zero, and refers to the register in the previous instruction. The loop, i.e. the sum and store process, is repeated until register B is zero (because all ten values have been read and summed) so, the jump to LOOP is not executed and the next instruction:

HLT

stops the program.

Now we understand the operation of

this program we are in a better position to examine how the assembler works. First, each instruction is read and each symbolic operand is placed in the symbol table; suitable values are then given to each symbol. For example, the instruction:

START

the first to be placed in the table, is assigned the value 0 showing that it represents the first memory location.

The assembler doesn't yet know the value of the operand ZERO (hence the need for two passes) so it simply places ZERO in the symbol table for later assignment; the symbol SUM is also placed into the symbol table without a value. The symbol LOOP is placed in the table and assigned the value 3 (the fourth memory location) as it is associated with the fourth instruction. (In this example we are assuming that each instruction requires one memory location.)

When the ZERO is finally read, the assembler knows what value to assign to this symbol, i.e. 9, the ninth memory location. Similarly, the value 10 is assigned to SUM.

The assembler now generates the object code during the second pass. As eachinstruction is read, the assembler calculates the correct machine code by changing each mnemonic instruction to its machine code op code and obtaining the value of each operand, either directly, or by looking them up in the symbol table.

The whole assembler process is illustrated in *figure 7*. The first pass of the assembler uses the memory location counter to keep track of the number of instructions and the memory locations. The symbol table keeps track of symbols and their values and the op code table lists the instruction mnemonics and their machine op code value. Any unidentified mnemonics or syntax errors are notified to the user; if there are none, the second pass produces the object code, along with a program listing. This listing usually provides both the assembler language instructions and the corresponding machine codes.

Although assembly language provides a method of programming at machine language level with a more easily understandable mnemonic notation, it is still difficult and time consuming, as the programmer needs to write every single instruction in source language. To alleviate this problem, **macro instructions** are utilised – these are single instructions which, when assembled, are replaced with many machine code instructions. These macros (some written by the manufacturer, and some by the user) are often used as a library of sub-routines to perform common tasks such as input/output operations. The assembler converts these macro instructions into object code.

ZEFA

Right: computers at work in a surveyors' office.

Compilers and interpreters

As we have said, a **compiler** is a program which translates high-level languages into machine language. These high-level language statements are more difficult to convert to machine code than assembly language because one instruction converts into a large number of machine instructions.

The compiler essentially does the following:

1) translates the source program statements ino machine code;
2) includes linkage for sub-routines;
3) allocates areas of main storage;
4) produces a printed listing of the source and object programs as required;
5) tabulates a list of errors found during compilation, e.g. use of 'words' or statements not included in the language vocabulary, or the violation of the syntax rules.

The process of compilation can be broken down into a number of stages (these are dependent on the type of compiler used) such as: lexical analysis, syntax analysis, transformation, intermediate language production and machine language production (the latter three could together be called code generation).

Lexical analysis

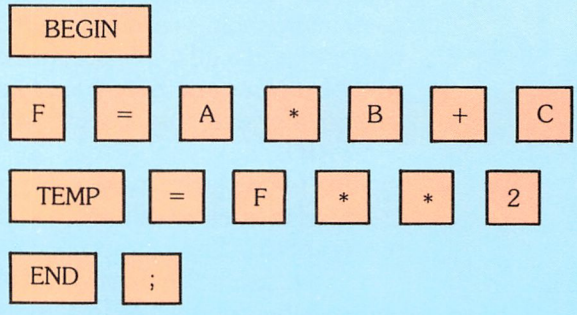
During this stage, the statements are scanned, character by character, and the fundamental parts or **tokens** are defined. (A token is the smallest meaningful representation of a concept in a language. It is a character group or construction that loses its meaning if divided.)

All statements are checked to ensure that they are valid, e.g. that data names are valid, and any irregularities in the format, e.g. a double decimal point, are noted. Formats are also standardised.

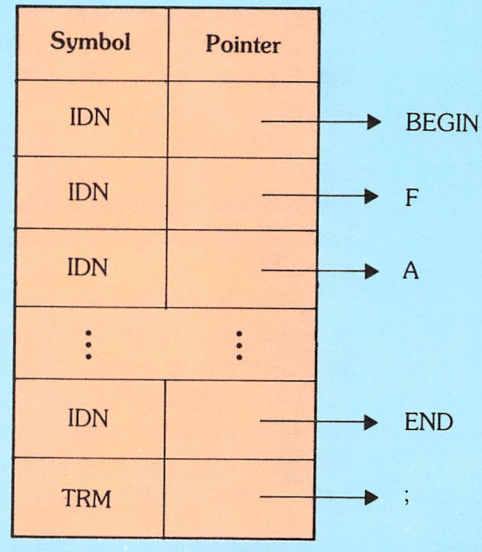
Syntax analysis

It is during this stage that the grammar, or syntax of the statements is checked (their meaning is not yet determined). Like the English language, words must be in their proper relationship with respect to each other in order for the statements to make 'sense'. Complex forms may also be broken down into more manageable forms.

8



9



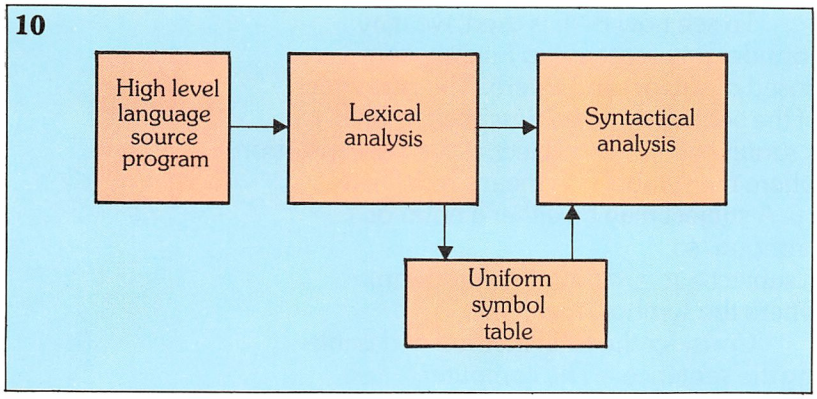
Code generation

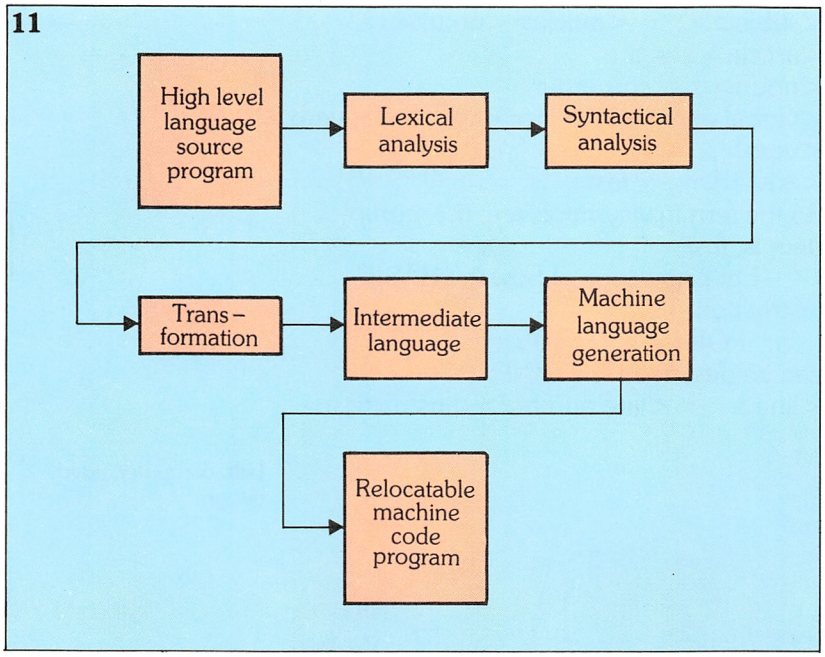
In the final stage of the compilation process, the variables and operators of the language statements are **transformed** into a tree format from which the **intermediate language** is constructed. (Remember, a tree is a linked list with multiple pointers that show the relationships between variables and operators.) This intermediate language code provides an optimised version of the program with respect to the facilities of the language. It is from this code that the compiler then generates the machine code.

8, 9, 10. Stages in lexical analysis.

11. A summary of the compilation process.

Right: a large computer installation.





ZEFA

Compiling a program

As we have seen, lexical analysis identifies the basic tokens of the source program. This process involves a simple string processing method where different kinds of variable strings are identified.

Consider the following block of statements:

```
BEGIN
F = A * B + C;
TEMP = F**2;
END;
```

The first step in lexical analysis is to identify the variables as illustrated in *figure 8*. During this identification, they are also classified and a **table of uniform symbols** for each class is constructed (*figures 9* and *10*). This table organises the variables in such a way that syntax checking and instruction interpretation are much simpler.

The number of characters per symbol which must be examined is always the same: for example, the symbol ; is classed as a terminal symbol and is given the name TRM; the variable A, on the other hand, is classed as an identifier (IDN). Each symbol is placed in the table in the order in which it is processed; all comments and blanks are extracted from the program statements.

The symbols table provides a **pointer** to the actual location of the symbol so that when the variable name is needed, the table provides a reference.

Transformation and the production of the intermediate language may be carried out with the use of a **stack**: you will remember that a stack is a last-in/first-out data structure.

The efficiency of a compiler depends primarily on the way in which those various tables are accessed and constructed.

A summary of the compilation process is shown in *figure 11*.

Interpreters

Interpreters, remember, are programs which translate source code a line at a time. This is not a particularly efficient method of translation, although interpreters can be useful in debugging programs as they run, and in handling software written for a different computer.

Translation techniques

Computer languages may be described in notational form by using a special language, such as **Backus Naur form** (BNF), named after its inventers John Backus and Peter Naur. BNF allows the syntax of whichever language the program is written in to be described in statements. Special languages which can do this to other languages are known as **meta-languages**. BNF, and other meta-languages, are used to describe the various computer languages in a form so that a compiler program can translate the instructions.

Like any language, BNF has certain rules and formats as follows. First, we need to distinguish between **terminal** and **non-terminal** symbols. The non-terminal symbols, sometimes known as **syntactic variables**, are enclosed in brackets and represent intermediate steps in the description process. The terminal symbols, sometimes known as **syntactic elements**, are the final symbols from which the generated description is eventually defined.

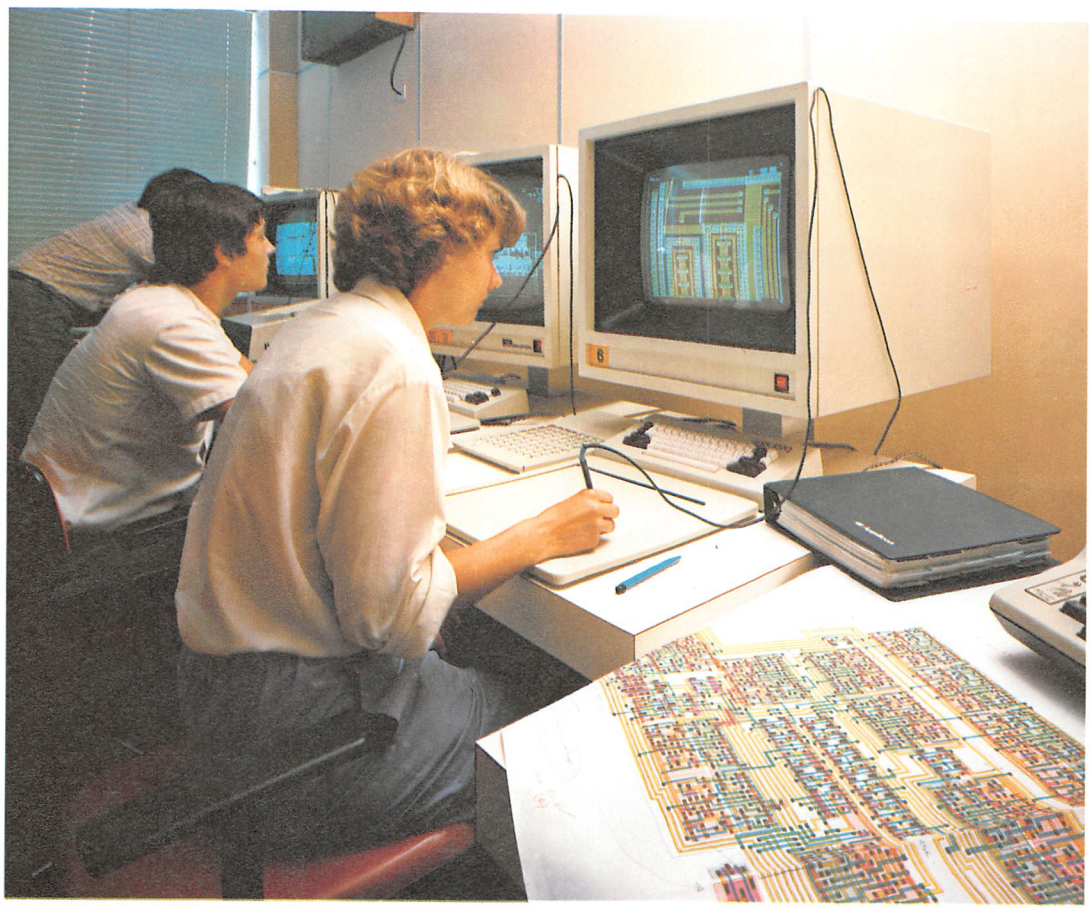
To see how BNF is used, we may consider any sentence in English, composed of subject and a verb. The grammar of the sentence can be described in BNF:
<sentence> :: = <subject><verbal expression>
where the symbol :: = means *is defined by*. A subject may be either a noun or pronoun, so:
<subject> :: = <noun>|<pronoun>
where the symbol | means *or*.

Consider the BNF method of describing the sentence – *The computer is fast*:
<sentence> :: = <subject><verbal expression>
<subject> :: = <article><noun>
<article> :: = the
<noun> :: = computer
<verbal expression> :: = <verb><adverb>
<verb> :: = is
<adverb> :: = fast
So the terminal symbols are: *the*, *computer*, *is*, *fast*.

Let's now look at how the BASIC instruction:

```
5  FOR I = 1 TO 10
```

can be described with BNF:
<line> :: = <line number><instruction>



The Research House/Marconi Electronic Devices Ltd

Left: computer aided design.

<line number> :: = <digit>
<digit> :: = 1|5|0
<digits> :: = <digit><digit>
<instruction> :: = FOR <variable>
= <digit> TO <digit>
<variable> :: = I

In this example we are dealing with only one BASIC statement which consists of a line number and an instruction; the line number consists of a digit. Because we are dealing with just one statement, a digit can only be 0, 1 or 5. The instruction consists of the key word FOR, the variable, the key word =, a digit, the key word TO, and two more digits. The <digit><digit> notation specifies the two digits that may be a combination of any numbers specified by <digit>.

As you can see, we have used BNF in this example to describe the grammar of a language with only one instruction. The description of an entire language obviously requires a much more lengthy set of specifications.

Reverse Polish notation

Reverse Polish notation (**RPN**) is useful for handling expressions where evaluation is governed by the rules of precedence. Invented by the Polish logician Lukasiewicz, it has the advantage that neither parenthesis nor operator priority is needed to determine the order in which the operations are to occur. For instance, with no operator priority, the expression:

a + b * c

may be mistaken for:

(a + b) * c

Similarly, the expression:

a + b/c * d

could be mistaken for:

(a + b)/c * d

There are several well defined algorithms for generating RPN which are based on the rule that an operator must always immediately follow its operands.

Taking the first example above, the + operator is to be applied to the operands a and b*c, so the first step in reconstructing the expression in RPN yields:

a b * c +

Second, we see that the multiplication symbol * involves the operands b and c so we get:

a b c * +

Similarly:

a – b * c + d

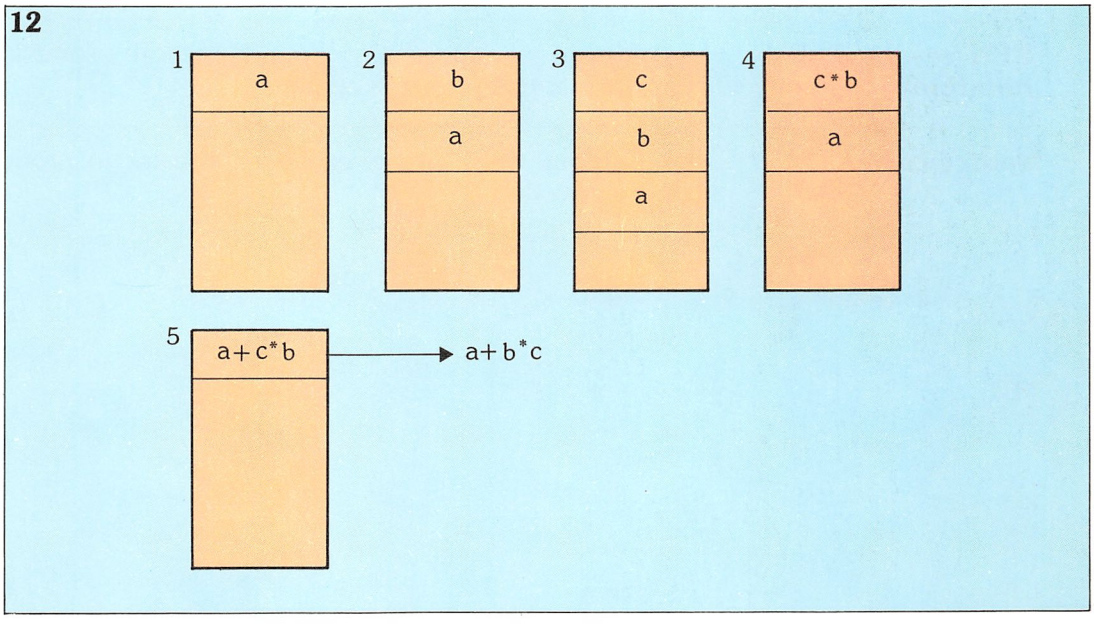
becomes:

a b c * – d +

The abc* – portion is almost the same as the first example. Since the + operator must follow its operands, the + must follow the d as shown.

As you can see, the operators are placed in the order in which they are dealt with, rather than in their conventional order.

Taking our first example, abc * +, we see that the RPN expression is evaluated



12. Expression evaluation from RPN.

by using a stack structure as shown in *figure 12*. First, variables a, b and c are pushed on top of the stack. Second, as the operator * is encountered, the two top elements, b and c, are popped from the stack, multiplied together and the result is pushed back onto the stack. Thirdly, the top two elements are popped from the stack, added together and the final result is pushed back on top of the stack.

Glossary

absolute address	the actual address of a location in store expressed in terms of the machine code numbering system
absolute loader	a program which loads absolute modules or programs into memory, starting with a specific location
BNF (Backus Naur form)	a method of describing the grammar of a language
cross translator	a program translator used on one computer which generates machine coded object programs for a different computer
linking loader	a program which loads relocatable modules into memory, and provides a means of linking them together, to generate absolute code for the total program
macro instruction	a single instruction written as part of the source program which, when assembled into machine code, generates several machine code instructions
object code	the machine coded output from a translator
pseudo instruction	an assembly language instruction which is not translated into machine code, but used to control the translation. Also known as a directive, or pseudo-operation
relocatable code	machine code which may be stored and executed in any part of the memory
RPN (reverse Polish notation)	a method of representing an arithmetical expression, where the operator is written after its operands
source code	instructions which are given to a translator to produce executable machine code